COMPUTER SCIENCE & TECHNOLOGY:

COMPUTER PERFORMANCE EVALUATION USERS GROUP

CPEUG
14th Meeting

NBS Special Publication 500-41
U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards

COMPUTER SCIENCE & TECHNOLOGY:

Computer Performance Evaluation Users Group (CPEUG)

Proceedings of the Fourteenth Meeting
held at Boston, Massachusetts
October 24-27, 1978

Editor:

James E. Weatherbee

Conference Host:

General Services Administration (Region 1)
Boston, Massachusetts and
the New England FIP Executive Council

Sponsored by
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

U.S. DEPARTMENT OF COMMERCE, Juanita M. Kreps, Secretary

Dr. Sidney Harman, Under Secretary
Jordan J. Baruch, Assistant Secretary for Science and Technology

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, Director

Issued October 1978

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

National Bureau of Standards Special Publication 500-41
Nat. Bur. Stand. (U.S.), Spec. Publ. 500-41, 353 pages (Oct. 1978)
CODEN: XNBSAV

Library of Congress Catalog Card Number: 78-600118

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1978

For sale by the Superintendent of Documents, U.S. Government Printing Office
Washington, D.C. 20402 - Price $6

Stock No. 003-003-01985-2

CPEUG 78

Foreword

As the number of computer performance-related conferences continues to grow, it seems timely to reflect on the qualities that have contributed to CPEUG's record of success in the field.

First, the CPEUG conference has always attempted to serve the widest possible audience without compromising on technical quality. At the extremes, tutorials are provided for newcomers, managers and analysts interested in broadening their expertise, while experienced practitioners meet in informal workshops to explore advanced problems of common interest. In between, CPEUG offers a rich and varied technical program and a distinguished conference proceedings. Although each conference has its own focal point, after fourteen conferences in seven years no one technique and no one vendor line of equipment has ever dominated a CPEUG program.

A second reason for CPEUG's success is its well-deserved reputation for stressing the practical side of performance technology. Yesterday's theories quickly become today's practice in performance evaluation, however, and as this year's program attests CPEUG always has room for promising new ideas and methods.

Finally, CPEUG is distinguished by its focus on the performance problems of one specific user of computers: the Federal Government, which, with annual ADP costs estimated at $15 billion, is the largest single consumer of computer power in the world today. CPEUG was founded by the Air Force in 1971 to help solve two chronic and characteristic problems of Federal data processing: competitive selection, and sizing for standard systems at multiple installations. Later, the National Bureau of Standards undertook the sponsorship of CPEUG because of its Brooks Act responsibilities to explore the applicability of standards to computer performance. If CPEUG has one defining characteristic, it is this continuing commitment to serve the most urgent requirements of Federal data processing.

Although Federal needs define the structure and direction of the CPEUG program, they exclude none of the important problems facing industry in its use of computers. Indeed the only difference sometimes appears to be that the Government encounters them first and lives with them longer. Consequently, better than half the papers and attendees at recent CPEUG meetings have been from the private sector. Last year's keynote speaker came to us from the plug-compatible peripherals industry, and this year's Program Chairman works for a major computer manufacturer. Clearly, Government and commercial data processors have much to learn from each other, and over the course of years CPEUG has become a meeting ground for the interchange.

We welcome all of you to CPEUG 78, and wish you an interesting and worthwhile experience while you are here.

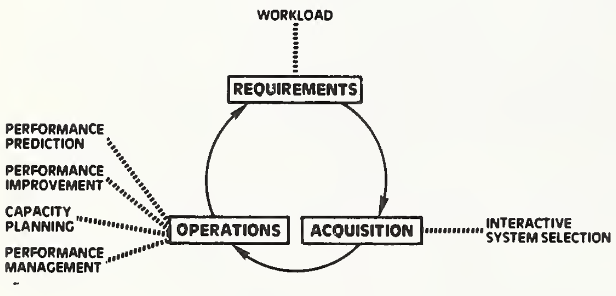
We would like to take special notice of the contributions of several people whose important services have helped to make this conference and this Proceedings possible. Our special thanks go to Pat Christoper (GSA Region 1), Fred Cross (GAO), Donna Edgerton (NAVORD), Caral Giammo (DCA/CCTC), Barbara Peterson (NBS) and Carol Zerr (FEDSIM).

Richard F. Dunlavey
Chairman, CPEUG
October, 1978

CPEUG 78

PREFACE

The technical program for CPEUG 78 extends the development of the ADP life-cycle theme introduced at last year's conference. The three fundamental life-cycle phases of primary interest in performance evaluation are: System Requirements, System Acquisition, and System Operations. The requirements phase poses the specific problem of estimating system load. Acquisition asks how to select the appropriate computer system. Operations requires us to anticipate the effect of changing loads and configuration, to tune or improve system performance, and to predict when the system will exceed its capacity. The following diagram shows the relationship between these major life-cycle phases:



This diagram is a model that helps us to define the dimensions of the computer performance space. It is presented here--and at the conference--for your review, comment, criticsm and refinement.

CPEUG regulars will notice several differences from last year's program. In the sessions pertaining to system acquisition, for example, there is more emphasis this year on tools and less on the policies and procedures of procurement. The sessions on performance measurement, the quantitative basis of CPE, emphasize methods for data analysis as opposed to techniques for data collection. New methods for predicting load, service and utilization, and highly quantitative techniques for performance improvement are introduced. And CPE has evidently advanced to the point that formal guidance documents on methodology are beginning to emerge, samples of which will be presented here.

Despite these changes, the informal tone of information exchange for which CPEUG is noted has been retained this year in the form of open post-session booths for performance vendors, related papers and birds-of-a-feather meetings. Finally, we have attempted to provide for the continued evolution of the conference and its sustaining theme by concluding with a workshop/forum on future directions in CPE.

We would personally like to thank the many people who have assisted us in making this conference a success. In particular, we would like to thank John Bennett and Carol Potter for the many hours they have contributed in organizing the post-session booths.

Terry W. Potter
Program Chairman

L. Arnold Johnson
Program Co-Chairman

CPEUG 78

ABSTRACT

The Proceedings record the papers that were presented at the Fourteenth Meeting of the Computer Performance Evaluation Users Group (CPEUG) held October 24-27, 1978 in Boston, Massachusetts. The program explored the technical and management issues of performance technology as they relate to the three major phases of the ADP life-cycle: Requirements, Acquisition, and Operations. The program included state-of-the-art papers and tutorials, a panel on the potential impact of ADP reorganization in the Federal Government, and a workshop on the future of CPE.

Key words: ADP life cycle; computer performance evaluation; computer performance measurement; computer performance prediction; computer system acquisition; conference proceedings; CPEUG; hardware monitoring, on-line system evaluation; prediction methods; queuing models; simulation; software monitoring; workload definition.

CPEUG78

CPEUG OFFICERS

° CHAIRMAN	Richard F. Dunlavey National Bureau of Standards Washington, DC
° VICE-CHAIRMAN	Gerald W. Findley General Services Administration Washington, DC
° SECRETARY/TREASURER	Dennis M. Gilbert FEDSIM/NA Washington, DC
° SPONSOR	John F. Wood National Bureau of Standards Washington, DC

CPEUG 78

PROGRAM COMMITTEE

° PROGRAM CHAIRMAN	Terry W. Potter Digital Equipment Corporation Maynard, MA
° PROGRAM CO-CHAIRMAN	L. Arnold Johnson Department of the Navy/FCCTS Washington, DC
° PUBLICATIONS CHAIRMAN	James E. Weatherbee FEDSIM/MV Washington, DC
° REGISTRATION CHAIRMAN	Margaret Maulin DOD Computer Institute Washington, DC
° PUBLICITY CHAIRMAN	Carol Wilson National Bureau of Standards Washington, DC
° FINANCE CHAIRMAN	Harry J. Mason, Jr. U.S. General Accounting Office Washington, DC
° ARRANGEMENTS CHAIRMAN	John Bongiovanni Air Force Data Services Center Washington, DC
° LOCAL HOST	Warren Patterson GSA Regional Office No. 1, 1CP Boston, MA

CPEUG78

CPEUG 78

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

CPEUG 78

SELECTION OF INTERACTIVE SYSTEMS: METHODS AND EXPERIENCES

SELECTION OF INTERACTIVE SYSTEMS: METHODS AND EXPERIENCES

Paul Oliver

Director
Federal COBOL Compiler Testing Service
Department of the Navy
Washington, D.C. 20376

Benchmark programs have for many years constituted an important component of technical evaluations preceeding the selection of data processing systems. Benchmarks have been used with varying degrees of success to model batch workloads, but have been severely limited in modeling time-sharing systems. Recent years have seen the development of new methodologies which are designed to reduce these limitations. The papers in this session examine applications of these methodologies to the measurement of the performance of time-sharing systems, new problems to be addressed, and upcoming regulations governing the use of remote terminal emulators in the Federal Government.

INCORPORATING REMOTE TERMINAL EMULATION INTO THE FEDERAL ADP PROCUREMENT PROCESS

Thomas F. Wyrick*

Directorate of System Evaluation
Federal Computer Performance Evaluation
and Simulation Center
Washington, DC 20330

and

Gerald W. Findley*

Office of Agency Services and Procurement
Automated Data and Telecommunications Service
General Services Administration
Washington, DC 20405

Remote terminal emulation is a benchmarking technique that can be used to validate the performance of teleprocessing (TP) systems or services for which it would be impractical to conduct a Live Test Demonstration with the total proposed network of computers, terminal devices, and data communications facilities. A Government program, now underway, is developing regulations and guidance documents that will (1) restrict when and how Federal agencies can use emulation during procurement and (2) describe the emulation benchmark capabilities that vendors must have to be qualified to bid on certain Federal TP procurements. The program is being conducted by the General Services Administration, Automated Data and Telecommunications Service, with assistance from both the Federal Computer Performance Evaluation and Simulation Center and the National Bureau of Standards, Institute of Computer Sciences and Technology. This paper describes the Government's program to incorporate remote terminal emulation into the Federal ADP procurement process and presents the program's background, objectives, schedule, and current status. Significant intermediate results, recommendations, and decisions are summarized. The paper also outlines the regulations and guidance documents to be produced. Selected technical and policy references are cited to encourage further investigation.

Key words: Procurement; selection; performance evaluation; benchmarking; remote terminal emulation; remote terminal emulators; teleprocessing.

---

*The views and conclusions contained in this paper are the authors' and should not be interpreted as representing the official opinions or policies of their organizations or of any other person(s) or agency associated with the Government. Moreover, this paper contains the authors' summaries of official documents and, therefore, may not reflect the full intent of those documents.

1. Introduction

Remote terminal emulation is a benchmarking technique that uses an external "driver" computer system to "stimulate" or impose teleprocessing (TP) workload demands on the ADP components, system, or service being tested (hereafter referred to as the System Under Test [SUT]). Many human operator and remote device (e.g., interactive, transaction, and batch terminals, concentrators, etc.) characteristics and actions can be represented precisely and in real time. Delays can be introduced to reflect propagation, modem turnaround, etc. The driver computer system can exchange control and application data transmissions with the SUT through the SUT's operational data communications hardware and software. Remote terminal emulation can use large numbers (up to several hundred) of data communications links of the same speeds, and with the same communications protocols, as an operational environment. When the remote terminal emulation technique is properly used, the SUT cannot distinguish if a real or emulated device is generating the workload. A monitor external to the SUT is a required component of the terminal emulation technique. The monitor records on a log file certain aspects of the interaction between the driver and the SUT. Such log files typically include all application data characters transmitted or received by an emulated device and the time each transmission was sent or received by the driver. Data reduction software produces various SUT performance measures (e.g., turnaround time, response time, etc.) from the log file after the test. A Remote Terminal Emulator (RTE) is a specific hardware and software implementation of such a driver system. A monitor is usually an integral part of an RTE.

The test workload elements for a benchmark test using remote terminal emulation must include descriptions of all the teleprocessing characteristics that are to be emulated, such as (1) the numbers and types of devices, (2) the character sets and protocols, (3) the number and speeds of communications links, (4) the assignments of devices to links, (5) all operator input, actions, pauses, and decisions, and (6) the system utilities, application systems, and data files accessed. For benchmark tests during multi-vendor procurements, these descriptions usually are in functional, vendor-independent terms, and are called scenarios.

RTE scripts are developed from the types of workload descriptions described above. Each script is a set of RTE instructions, data, and procedures that causes a particular RTE to impose specific test workload demands on a given SUT. The set of scripts comprises the total test workload imposed by the RTE during a benchmark test.

Remote terminal emulation can be used to satisfy many performance evaluation objectives, such as regression testing, stress load analysis, system integration, migration planning, procurement evaluation, etc. Though originally developed for testing large host systems, this technique has been used successfully by both vendors and users to test large and small hosts, front-end communications processors, message switches, intelligent terminal systems, etc. The reader should refer to references 1 and 2 for more detailed discussions of teleprocessing system benchmark test objectives, techniques, and remote terminal emulation.

In 1976, the US Government began a joint Government-Industry project to incorporate the use of remote terminal emulation in Federal ADP procurements. Several factors led to the initiation of this project:

1) the increasing Government requirements for effective and efficient teleprocessing support;

2) the importance that the Government places on reducing cost risks and mission risks by validating, before contract award, vendor claims of performance;

3) the limited comparability of emulation benchmark test results that can be caused by a lack of functional similarity between RTE's;

4) the possible limiting effects of remote terminal emulation on free and open competition; and

5) the expense, both to Government and industry, of using RTE's for procurement evaluation.

The project is being conducted by the General Services Administration, Automated Data and Telecommunications Service (GSA/ADTS), with assistance from both the Federal Computer Performance Evaluation and Simulation Center (FEDSIM) and the National Bureau of Standards, Institute for Computer Sciences and Technology (NBS/ICST). Mr Gerald W. Findley, Director of the Special Projects Staff in the Office of Agency Services and Procurement, GSA/ADTS, is Project Director. Mr Thomas F. Wyrick of the Directorate of System Evaluation, FEDSIM, directs FEDSIM's involvement. Dr Marshall D. Abrams of the Computer Networking Section, NBS/ICST, coordinates NBS's participation.

The project consists of a completed feasibility and requirements study phase and an ongoing guidance development phase.

2. Feasibility and Requirements

In April 1976, the Government began a project phase to analyze the feasibility of, and requirements for, using remote terminal emulation in Government ADP procurements [3][1]. A principal feature of this phase was the close and continuing interaction with both Government and industry. The phase addressed both TP system and service (e.g., remote time-sharing service, etc.) procurements, and contained six analysis tasks. These tasks were:

1) Specification of Teleprocessing Requirements - Future Agency Needs

   Task personnel examined TP workload definition and specification techniques, apparent workload trends, the impact of these trends on future specification techniques, and alternate methods for specifying teleprocessing benchmark tests [4].

2) Survey of Techniques for Emulating Teleprocessing Workloads

   Task personnel surveyed remote terminal emulators, other (i.e., non-RTE) techniques for imposing TP workloads during benchmark tests, and certification and verification issues related to each technique [2,5].

3) GSA/NBS Workshops on Remote Terminal Emulation

   Task members conducted a RTE workshop for Government personnel and a workshop for both Government and industry personnel [6,7].

4) Government ADP Procurement Projection

   Task members projected the quantities and costs of TP systems and services that the Government will procure into the 1980's, the types and quantities of remote TP devices that will be on-line, and the types and relative importance of the TP applications that will be used [8].

5) Procurement Validation Case Studies

   Task personnel surveyed TP procurement, performance validation, and benchmarking concepts, experiences, and opinions of ADP professionals, in and out of Government, to help determine whether, and under what circumstances, remote terminal emulation should be used in Federal ADP procurements [9].

6) Requirements Analysis and Development of Recommendations

   Task members consolidated the findings and results of the other project tasks, presented alternatives for incorporating emulation into Federal procurements, and recommended whether or not emulation should be used in future Government procurements [10].

The principal findings of this project phase fall into three categories and are summarized below.

2.1 General Findings

Almost all major system and service vendors have remote terminal emulation capabilities and use these capabilities for internal corporate studies not related to procurement evaluation. If emulation were regularly used in Government procurements, then both the procuring agencies' selection costs and vendors' proposal submission costs (i.e., vendors' "entry fees") would be higher than most procurements where emulation were not used. Competition probably would be reduced if emulation were used regularly, because only vendors with emulation capabilities would be able to bid on those Federal procurements that would require emulation.

Vendors and Government agencies expressed a desire for Government-wide guidance documents which suggested when and how agencies should use remote terminal emulation and described the functional emulation benchmark test capabilities that vendors could be required to provide.

2.2 System Procurement Findings

There is no technically valid, fair, equitable and generally applicable alternative

[1]Figures in brackets indicate the literature references at the end of this paper.

to remote terminal emulation for validating, before contract award, the performance of vendor-proposed TP systems that support large numbers of remote devices. The Government will procure a significant and growing number of such TP systems during the next few years; it was estimated that a minimum of 42 such systems will be procured during Fiscal Years (FY's) 78 and 79, and that at least 68 systems will be procured during FY's 80 and 81 [8]. Some Government agencies had used, and others were planning to use, emulation for TP system procurement evaluation. Several commercial organizations also had used emulation for procurement evaluation [11].

2.3 Service Procurement Findings

The general applicability of remote terminal emulation for procurement evaluation of TP services has not been clearly established. An agency procuring services usually does not acquire a vendor's total ADP capacity, and the winning vendor usually can change his total ADP capacity after contract award. A multi-terminal benchmark test that evaluates a service vendor's total ADP capacity, therefore, is of questionable value. Alternatives to remote terminal emulation are available for representing the loads imposed by small numbers of terminals. No Government agencies were identified that had used, or were planning to use, emulation during TP service procurements. Only one commercial organization was identified that had used an RTE for procurement evaluation; that organization used the RTE to represent only one terminal. No commercial groups were found that planned to use emulation during future service procurements.

2.4 Recommendations

In October 1977, the project team prepared a draft GSA Decision Paper that included recommendations about the future of remote terminal emulation in Federal ADP procurements. For TP system procurements, we recommended that GSA allow Federal agencies to require vendors to provide emulation benchmark capabilities, and that GSA publish guidance describing when and how emulation should be used. Each agency would decide whether or not to use remote terminal emulation during each of its TP system procurements, based upon its interpretations of such factors as mission risks and procurement preparation costs. If an agency determined that emulation were necessary for procurement evaluation, then the agency would disqualify any vendors that did not provide the necessary emulation benchmark test capabilities.

For TP service procurements, we recommended that GSA forbid Federal agencies to use remote terminal emulation, and that GSA conduct a separate study of cost-effective techniques for procurement evaluation of TP services. GSA would consider individual exceptions to this service procurement ban for agencies with unusually large and complex TP requirements.

Mr Frank J. Carr, Commissioner of GSA/ADTS, approved all project recommendations on January 20, 1978, after many Government and industry organizations had reviewed and commented on the draft Decision Paper [10]. Commissioner Carr also authorized the preparation of Federal Procurement Regulations (FPR's) to implement these recommendations, and a project to develop the remote terminal emulation guidance [12].

3. Guidance Development

The guidance will be organized into two documents:

1) "Use of Remote Terminal Emulation in Federal ADP System Procurements," and

2) "Remote Terminal Emulation Specifications for Federal ADP System Procurements."

The documents will be published and distributed by GSA/ADTS and will explain how to interpret and comply with the FPR's on this subject. They will not be part of the Federal Information Processing Standards Publication (FIPS PUB) series and will not themselves be regulations. They will be referenced by name, however, in certain FPR's.

The document entitled "Use of Remote Terminal Emulation in Federal ADP System Procurements" will summarize relevant procurement evaluation, benchmarking, and remote terminal emulation concepts and terminology, and will describe the scope and applicability of both guidance documents. It also will:

1) present some of the factors and criteria that agencies should consider when deciding whether to use remote terminal emulation;

2) outline the types of analyses that should be made during the design and development of emulation benchmark tests;

3) suggest operational procedures for using emulation, including the items to be supplied and functions to be performed by Government agencies and system vendors;

4) define the data elements and formats an agency should use to describe TP workloads for emulation benchmark tests;

5) recommend a glossary of relevant terminology;

6) include a bibliography of significant technical and policy materials; and

7) cite sources of intragovernment assistance (e.g., FEDSIM, NBS/ICST, etc.)

This document will be as similar in structure as possible to "Guidelines for Benchmarking ADP Systems in the Competitive Procurement Environment," FIPS PUB 42-1 [13], and will use similar terminology whenever possible. The emulation guidance document will supplement, not rewrite, FIPS PUB 42-1 in the area of remote terminal emulation benchmark tests.

The document entitled "Remote Terminal Emulation Specifications for Federal ADP System Procurements" will define functional emulation capabilities, RTE log data reduction capabilities, and RTE log tape formats that should be common among vendors. This commonality will insure that TP benchmark test results will be more comparable, fair, and equitable than such results have been in the past. A Government agency will be able to require ADP system vendors to provide RTE's and test facilities that meet the specific portions of the specifications that are needed by that agency for procurement evaluation. Vendors without these capabilities can be disqualified from any Federal procurement where an agency has determined that these capabilities are required to validate the performance of proposed systems. Government agencies will be prohibited, however, from requiring vendors to provide capabilities not included in these specifications, except under extremely unusual circumstances. The specifications, therefore, will be a compromise between vendors' current emulation capabilities and agencies' projections of future emulation benchmark test requirements.

In addition to developing the two guidance documents, the project team will also recommend mechanisms to periodically review and revise the documents as needed. Our goal is to produce initial versions of the documents that are realistic, useful, and timely. The documents should be revised regularly to reflect changing TP technology, as well as to incorporate additional practical experiences gained through increased Government and industry use of emulation.

3.1 Project Schedule

Table 1 summarizes the guidance development project schedule. The project began in March 1978. Between April 19 and June 7, project personnel met with most major ADP system vendors to (1) inform them of the objectives, structure, and schedule of the project, (2) receive vendors' assessments of the desirability of, and difficulty they would have, implementing certain emulation capabilities, and (3) discuss the vendors' current and planned emulation capabilities. We met with the Government Interagency Committee on ADP (IAC/ADP) on May 9 to describe the project and to solicit agency opinions and requirements for emulation benchmark test capabilities. Because the specifications developed by this project might force some vendors to redesign their RTE's, we will meet again with ADP vendors in late September to discuss the preliminary results of our analysis of functional emulation capabilities.

Table 1. Guidance Development Project Schedule

1978	
Mar 1	Project Initiation
Apr 19	Begin Initial Vendor Meetings
May 9	IAC/ADP Meeting
Jun 7	Complete Initial Vendor Meetings
*Late Sep	Second Vendor Meetings
Oct 24	Release Draft Documents
Oct 24-27	CPEUG Conference
Dec 14	Public Workshop
1979	
Mar 1	Issue Final Documents Recommend Document Maintenance Mechanisms (Project Completion)
Jun 1	Guidance Effective

*Tentative

Drafts of both guidance documents are scheduled to be released for Government and industry review on October 24, 1978. Copies will be available from

Mr. Gerald W. Findley
GSA/ADTS/CDD
18th and F Sts., NW, Rm G241B
Washington, DC 20405
Telephone (202) 566-1076

The project team also plans to distribute copies at the Computer Performance Evaluation Users Group (CPEUG) Conference, October 24-27, in Boston, MA. On December 14, we will hold a public workshop in Washington, DC, to discuss the drafts. The documents will be revised as necessary, based upon the comments received, and will be issued in final form about March 1, 1978. On March 1, we will also recommend mechanisms for the periodic review and revision of the documents. On June 1, 1979, the guidance is scheduled to take effect. At that time, agencies may require vendors to provide the emulation capabilities specified in the documents.

3.2 Status

At present (August 1978), the project team is analyzing the data gathered from Government agencies and vendors and is developing detailed outlines for both documents. Many usage topics and functional emulation specifications are being examined for possible inclusion in the guidance documents, including the examples summarized below. For more information, the reader should refer to the draft guidance documents, scheduled to be released at the time this paper is published.

Currently we are examining four broad remote terminal emulation usage topics:

1) Teleprocessing Workload Test Descriptions,

2) RTE Verification Techniques,

3) Vendor-provided Benchmark Test Documentation, and

4) Benchmark Test Reruns after Installation.

The guidance documents will recommend the data elements and formats that agencies should use to describe and interrelate at least four basic TP workload components: (1) terminal operator actions (e.g., input, output, decisions, rates, etc.); (2) SUT-network interface characteristics (e.g., terminals, lines, protocols, etc.); (3) the software with which the operators interact (e.g., vendor-proposed text editors, Government-supplied applications, etc.); and (4) the data files accessed and/or created by the operators. The general relationship of these four components is shown in Figure 1. We also are developing generalized RTE log file reports. Agencies often require vendors to provide documentation that varies from procurement to procurement; vendors, therefore, expend significant resources modifying report contents and formats to satisfy individual agencies. Generalized RTE log file reports will save vendor and Government resources if the reports satisfy most agency requirements. In addition, we are developing realistic procedures to verify that the RTE portions of a benchmark test proceeded according to specifications. Verification procedures being considered include (1) broadcasting Government-specified messages, at Government-selected times, from the SUT console to all or selected emulated terminals, (2) using a data communications line monitor, provided by vendors and/or the Government, to display and record all transmissions over Government-selected communications cables between the SUT and the RTE, and (3) changing, immediately prior to benchmark test execution, the contents of selected data files accessed by emulated terminals.

The project team is analyzing six preliminary categories of functional specifications:

1) Remote Device Representations,
2) Terminal Operator Representations,
3) Data Communications Link Representations,
4) RTE Driver Characteristics,
5) RTE Monitor Characteristics, and
6) RTE Log Analysis.

In the category of remote device representations, we are evaluating whether to specify the types and maximum numbers of devices that the Government could require vendors to emulate. These devices could be specified by generic type (e.g., asynchronous teleprinter, etc.) and/or by specific make and model (e.g., Teletype Model 33, etc.); the choice of make and model could be based upon the large Government inventory of owned terminals. Two other possible functional specifications under evaluation are the terminal character sets and transmission protocols that could be emulated for a benchmark test. The guidance documents may specify that certain Federal Information Processing Standards, American National Standards Institute (ANSI), and widely-used character sets and protocols be emulated. We also are considering whether

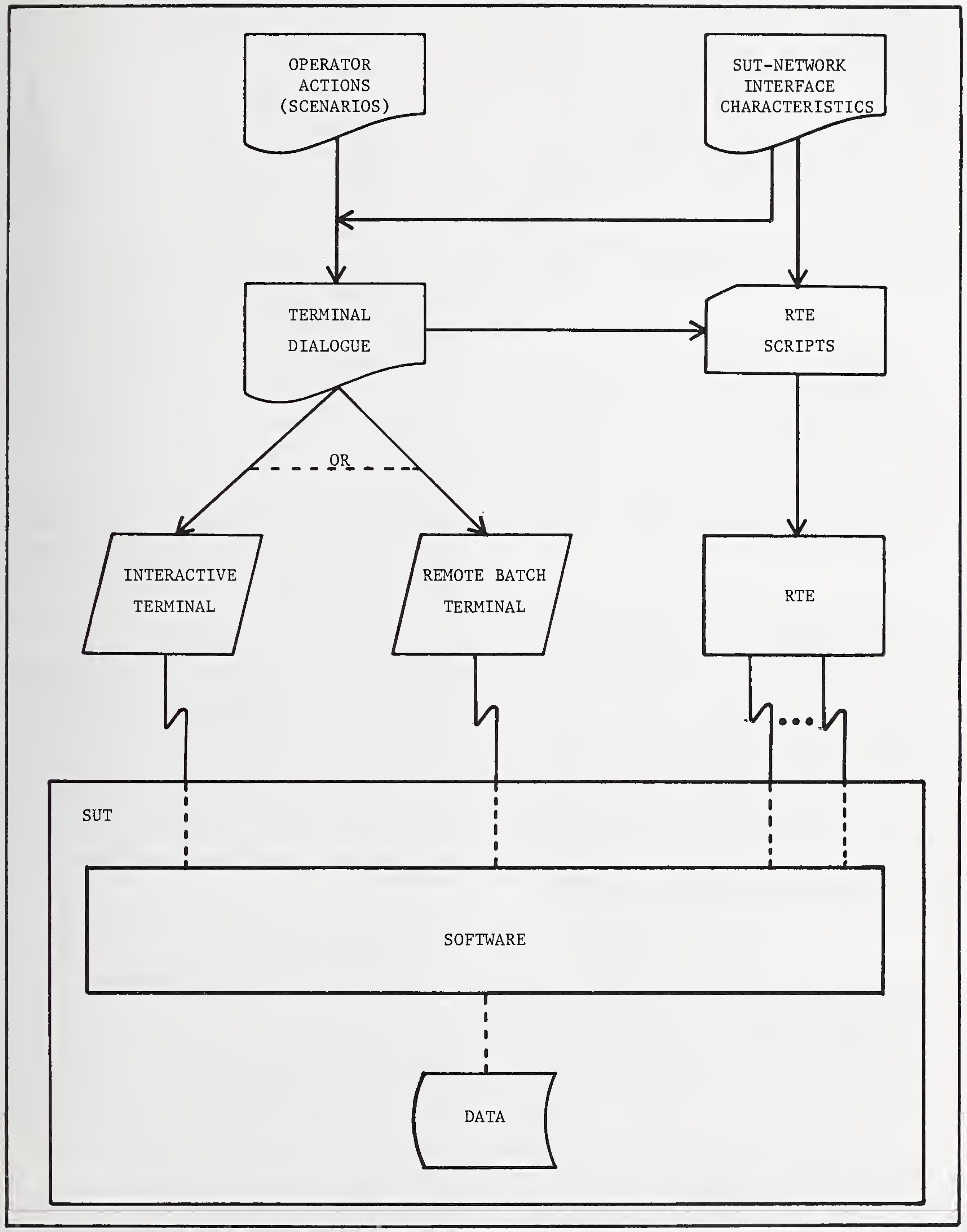


Figure 1. TP Workload Components for Benchmark Tests Using Remote Terminal Emulation

to specify the maximum number of data communications cables that the Government could require a vendor to configure between an RTE and the SUT. Some vendors have installed emulation facilities that permit them to easily configure several hundred cables; other vendors have never configured over forty cables, and agencies' requirements vary widely. Vendor costs to install and maintain hundreds of cables are often high, but the Government needs a reasonable number of cables to represent planned terminal and line loads. Any specification, therefore, will be a compromise.

4. Procurement Policy and Regulations

As of August 1978, the Government has no formal regulations on the use of remote terminal emulation during Federal ADP system and service procurements. GSA/ADTS, however, does have an interim policy and procedures covering emulation in systems procurements only [14]. The interim policy and procedures were developed jointly with industry during 1976 and are summarized below.

4.1 Interim Policy

Solicitation documents shall not be structured in such a way as to require offerors/bidders either to provide or to use RTE's.

Use of remote terminal emulation shall not be made mandatory in solicitation documents for Live Test Demonstrations (LTD's) or benchmark tests of ADP systems, unless it has been determined that emulation is the only practicable means of measuring teleprocessing performance. Under no circumstances shall the use of vendor-supplied RTE's be made mandatory for installation acceptance testing.

4.2 Interim Procedures

Remote terminal emulation and alternative methods for determining acceptable teleprocessing performance shall be considered and evaluated as to applicability to specific situations before specifying the use of any approach or combination of approaches in a specific ADP system procurement.

If it has been determined that there is no acceptable alternative for measuring teleprocessing performance other than the use of an RTE during the LTD, an RTE may be specified as mandatory in solicitation documents, provided:

1) Industry is given notice 30 days prior to release of the solicitation. This advance notice shall consist of pre-release of the detailed LTD instructions specifying the exact manner in which the RTE is to be used; notice of availability of these instructions shall be published in the Commerce Business Daily; and

2) GSA is provided detailed justification for use of the RTE and all related Request for Proposals (RFP) provisions at the time of industry notice.

4.3 Temporary Federal Procurement Regulation

GSA has drafted a Temporary Federal Procurement Regulation (FPR) on the use of emulation in both TP system and service procurements [15]. As of August 1978, the Temporary FPR is under final review at GSA and probably will be issued by the time this paper is published. (The reader can contact GSA to obtain the latest regulations, policies, and procedures.)

For system procurements, the Temporary FPR specifies that emulation benchmarks shall not be made mandatory in solicitation documents unless it has been determined that emulation is the only practicable means of measuring teleprocessing performance. Under no circumstances shall the use of a vendor-supplied RTE be made mandatory for installation acceptance testing. For service procurements, the FPR specifies that, pending further study by GSA, the mandatory use of remote terminal emulation is prohibited with the exception of:

1) dedicated teleprocessing requirements, and
2) unusually large and complex shared teleprocessing requirements.

The Temporary FPR further requires Federal agencies to evaluate remote terminal emulation and other performance evaluation methods before specifying the use of any approach or combination of approaches in a specific system or service procurement. If an agency determines that emulation is necessary in a given system procurement, then the agency must follow procedures identical to the interim procedures described above. Agencies that want to require emulation in a specific service procurement must identify and justify the specific exception sought and must provide this information to GSA.

5. Conclusion

The Government's remote terminal emulation project is a significant undertaking. The regulations and supporting guidance documents will affect vendors' benchmark test facilities and RTE's, and the way Government agencies perform benchmark tests during future TP system and service procurements. While the results of this project will help emulation benchmark tests to be more comparable, fair, and equitable than in the past, such tests will remain difficult and expensive to conduct well. The references cited below provide more information on procurement evaluation, benchmarking, and remote terminal emulation.

The authors thank the many ADP professionals, in both Government and commercial organizations, who have freely shared with the emulation project team their procurement evaluation and benchmarking experiences.

References

[1] Wyrick, T. F., Benchmarking Distributed Systems: Objectives and Techniques, Performance of Computer Installations, ed. D. Ferrari, North-Holland, New York, 1978.

[2] Watkins, S. W., and Abrams, M. D., Survey of Remote Terminal Emulators, Special Publication 500-4, National Bureau of Standards, Washington, DC, April 1977.

[3] General Services Administration, GSA's Project Plan for Incorporating the Use of Remote Terminal Emulation in the Federal ADP Procurement Process, GSA/ADTS, Washington, DC, April 1976.

[4] Conti, D. M., Specification of Teleprocessing Requirements - Future Agency Needs: Federal Agency Practice and Needs in the Area of Teleprocessing Workload Specification, Report CS77-2, GSA/ADTS, Washington, DC, October 1976.

[5] Abrams, M. D., and Watkins, S. W., Summary of Findings on Alternates to Remote Terminal Emulation For Imposition of Teleprocessing Workloads and Integrity Confirmation Aspects of Emulating Teleprocessing Workloads, Report CS77-4, GSA/ADTS, Washington, DC, November 1976.

[6] General Services Administration, Summary of the NBS/GSA Government Workshop on Remote Terminal Emulation, Report CS76-1, GSA/ADTS, Washington, DC, June 1976.

[7] General Services Administration, Summary of the NBS/GSA Public Workshop on Remote Terminal Emulation, Report CS76-2, GSA/ADTS, Washington, DC, September 1976.

[8] Wyrick, T. F., and Ball, C. G., Government ADP Procurement Projection, Report CS77-1, GSA/ADTS, Washington, DC, October 1976.

[9] Wyrick, T. F., Procurement Validation Case Studies: Concepts and Issues Relevant to the Use of Remote Terminal Emulation in Teleprocessing Procurements, Report CS77-6, GSA/ADTS, Washington, DC, May 1977.

[10] General Services Administration, Decision Paper: Should The Remote Terminal Emulation Benchmark Technique be Used in Future Federal ADP Procurements?, GSA/ADTS, Washington, DC, January 1978.

[11] Mukherjee, A., and Lauro, J., An Improved Benchmark Performance Evaluation Technique for Vendor Selection Studies, 1976 CPEUG Conference Proceedings, Washington, DC, November 1976, pp. 192-195.

[12] General Services Administration, Development of Remote Terminal Emulation Guidance Documents for Federal ADP System Procurements, GSA/ADTS, Washington, DC, January 1978.

[13] Guidelines for Benchmarking ADP Systems in the Competitive Procurement Environment, FIPS Publication 42-1, NBS, Washington, DC, May 1977.

[14] General Services Administration, Use of Remote Terminal Emulation, ADTS Operating Policies and Procedures for Procurement and Associated Program Direction, GSA Handbook DTS-P-2800.1, GSA/ADTS, Washington, DC, March, 1978, pp. 4.1-4.5.

[15] General Services Administration, Use of Benchmarks and Remote Terminal Emulation for Live Test Demonstrations of ADP Systems and Services, Draft Temporary Federal Procurement Regulation, GSA, Washington, DC, May 1978.

[16] Arthur, C. T., Remote Terminal Emulator Development and Application Criteria, 1977 National Computer Conference, AFIPS Conference Proceedings, Montvale, NY, 1977, pp. 733-739.

[17] McFaul, E. J., Application of Remote Terminal Emulation in the Procurement Process, 1977 National Computer Conference, AFIPS Conference Proceedings, Montvale, NJ, 1977, pp. 729-732.

[18] Computer and Business Equipment Manufacturers Associations, Use of Remote Terminal Emulation in ADP Procurement to Validate the Performance of Large Terminal Oriented Systems, Summary of the NBS/GSA Public Workshop on Remote Terminal Emulation, Report CS76-2, GSA/ADTS, Washington, DC, September 1976, pp. 177-182.

[19] Walkowicz, J., Benchmarking and Workload Definition: A Selected Bibliography with Abstracts, Special Publication 405, NBS, Washington, DC, November 1974.

APPLICATION OF A NETWORK MONITOR TO THE SELECTION OF A TIME SHARED COMPUTING SYSTEM

Marshall D. Abrams

Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, DC 20234

H. Philip Hayden

Advanced Systems Development Group
David W. Taylor Naval Ship Research and Development Center
Bethesda, MD 20084

This paper discusses the use of network service measurement techniques in the evaluation of system responses as part of the competitive selection of a multi-user time shared computing system for David Taylor Naval Research and Development Center (DTNSRDC). The selection of measured system responses was based on the assumption that there would be a direct relationship between the responses and the "quality of service" to the system's user community.

Key words: Benchmark; computer performance evaluation; Network Measurement System; response; service quality; selection.

1.0 INTRODUCTION

This paper conveys the authors' experiences during the procurement of a computer system for the David Taylor Naval Ship Research and Development Center (DTNSRDC). It begins with a brief summary of the available methodologies applicable to such an exercise, goes on to describe the requirements of this procurement, and terminates with a summary of the performance exhibited by the vendor's system to which award was made.

The complexities of modern computer systems, along with the economic necessity to maximize their utility, have led to the development of new techniques with which to evaluate computer system performance. While computer performance evaluation (CPE)

Contribution of the U. S. Government; not subject to copyright. Commercial products and descriptive data are provided for identification and exemplary purposes only. In no case does such identification imply recommendation, endorsement, or representativeness; nor does it imply that the products identified are the best available for the purposes described.

techniques have always included throughput and response time as important measures, they have focused primarily on computer system hardware measures such as cycle time, instruction execution time(s), and memory access time to provide a description of the computational power of the computer. These hardware-oriented measures were adequate for first and second generation single-user, batch processing systems. Such measures have very little meaning when applied to third generation multi-user, time-sharing systems in which the direct relationship between hardware and software is obscured by factors such as multi-programming and virtual memory. Modern CPE techniques have begun to focus on measures of performance which relate to the capability of the system to deliver services to users, rather than on the raw computational power of the hardware. User service measures include throughput (a measure of the total number of user service requests which can be satisfied in a given unit of time) and responsiveness (a measure of the time required to satisfy a single service request). Throughput is the measure primarily applied to batch systems while responsiveness is the measure primarily applied to time-sharing systems.

The CPE effort described in this paper is directed toward providing quantitative answers to questions concerning hardware, software, and workload. Typical questions include:

1. What is the maximum number of time-sharing users that can be supported by a given hardware/software configuration?

2. How sensitive is computer performance to changes in workload? Are there workload threshold levels beyond which performance is highly sensitive or insensitive?

3. How can scheduling parameters be adjusted so as to provide acceptable throughput for batch users while at the same time providing acceptable response for time-sharing users?

These types of questions continually arise during the course of activities associated with computer performance projection, computer selection evaluation, and computer performance optimization.

2.0 BENCHMARK COMPONENT OF A SPECIFIC PROCUREMENT

2.1 The Benchmark Test

A benchmark test [3] was formulated as part of the effort to select, via competitive procurement, a conversational processor for the DTNSRDC Advanced Computer Facility (ACF). This test was specifically directed toward verifying that the conversational processors proposed would have sufficient resources and scheduling capabilities to satisfy the response time requirements set forth in the ACF technical specifications.

The benchmark scenario was carefully constructed so as to be representative of the workload projected for the operational use of the conversational processor. As used here, "representative" means that the benchmark workload was generated to closely represent the real workload; data were collected from actual utilization time periods and from real jobs that were thought to cover a broad range of important operating conditions. The job mix in this projected workload consisted almost exclusively of interactive job streams in which job definition and decisions affecting job processing were dynamically specified by the users during job execution.

Response time was identified as the primary measure to be employed in judging the acceptability of the proposed system(s). Two qualifications were necessary in order to make response time a useful measure. Response time had to be precisely defined, and the action to which response was being made (and measured) had to be specified, as detailed below.

As previously identified [2], there are many extant definitions of response time. The definition recommended by NBS [4] is the elapsed time from the input of the last character (keystroke) from the conversational terminal to the conversational processor until the display of the first character of meaningful output (output indicating that the service request has been answered) at the conversational terminal.

Response time goals were based on two important assumptions:

1. Acceptable response can only be determined in terms of the reasonable expectations of the users. The numerical values specified were based on service levels on the interactive system then in use at DTNSRDC which the users identified as satisfactory and unsatisfactory.

2. The response time for specific requests should depend more strongly on the resources required to service the request than on the load on the system at the time the request was being serviced, i.e., the response time should be independent of the total workload mix which the computer was processing.

The goals gave some indication as to what may reasonably be expected from the system. Implicit in these goals was the desire for consistency in the responsiveness of the system to the same set of commands.

It is also well known [5] that the acceptability of a given response time is dependent on the activity being performed by the terminal user to which response is being made. DTNSRDC identified seven types of user activity and specified acceptable response times for each. These activities are described below and summarized in Table I.

Table I. Benchmark Functions

FUNCTION	MAXIMUM ACCEPTABLE RESPONSE TIME(SEC.)
1. Log-on	6.0
2. EOM to CR	0.2
3. Character echo	0.2
4. Character backspace	0.2
5. Line delete	0.2
6. File attach	3.0
7. Output abort	1.0
8. File status query	3.0
9. File save	3.0
10. File purge	3.0
11. Access to subsystem	3.0
12. Intra-line editing	3.0
13. Batch job submittal	3.0
14. Job status query	1.0
15. Load and execute small program	6.0
16. Load and execute medium program	20.0
17. Load and execute large program	60.0
18. Load and execute CPU-bound program	120.0
19. Task abort	1.0
20. Log-out	6.0

The specific job mix that was used to evaluate system response was explicitly described as part of the benchmark. The maximum response time requirements are as follows:

a. Motor feedback. Less than .2 seconds response for the following:

(1) Echoing a typed character back to the conversational terminal when operating in full duplex mode.

(2) Positioning of the cursor to a new line following the depression of an end-of-message key.

(3) Performing the control function to backspace a character.

(4) Performing the control function to erase a line of characters.

Approximately 10% of the logged-on users issued motor feedback response service requests.

b. Instantaneous Response. Less than 1 second response is required for the following:

(1) Inquiry respecting the status of a job

(2) Task abort

(3) Output abort

Approximately 5% of the logged-on users issued instantaneous response service requests.

c. Fast Response. Less than three seconds response is required for the following:

(1) Inquiry respecting the names of files currently attached to a user job.

(2) Attaching a file to a user job.

(3) Purging a file.

(4) Save a file.

(5) Accessing a subsystem.

(6) Each individual procedure required to log on and validate a user for access to the system.

(7) Log out.

(8) Intra-line editing.

(9) Acknowledgement of the submission of a batch job.

Approximately 55% of the logged-on users issued fast response service requests.

d. Quick Response. Less than six seconds response for loading and executing a small user program. The CPU and central memory requirements (expressed here in terms of CDC 6600 units which were familiar to the people who prepared the specifications) for executing a small user program are projected as normal distributions with means of .3 CPU seconds and 40K (octal) central memory words.

Table II. Background Job Streams

The scripts which specify the background job streams are listed herein. The number of users per script is specified.

A. <u>Script 1</u>

<u>Commands</u>

1. "LOGON" with password "MF,NSRDC".
2. Type in the following text 20 times: "The quick and crazy red and brown fox jumped over the little fence."
3. Type in the text "HERE" and then perform the "LINE DELETE" function. Repeat this exercise 6 times.
4. Type in the text "ANTIDISESTABLISHMENTARIANISM" 2 times and then perform "CHARACTER BACKSPACE" function 2 times.
5. "LOGOUT".
6. Repeat 1 through 5 as required for continuous execution during the benchmark test.

There shall be at least 5 users executing this script. The time interval between the initiation of a LOGON command and the completion of the LOGOUT command by each user shall not exceed 361 seconds.

B. <u>Script 2</u>

<u>Commands</u>

1. "LOGON" with password "INST,NSRDC".
2. "ATTACH" file whose file name is "BENCH 1".
3. Perform function to "LIST" file at user terminal and then perform the "OUTPUT ABORT" function.
4. "QUERY" the system for the status of this job.
5. "LOGOUT".
6. Repeat 1 through 5 as required for continuous execution during the benchmark test.

There shall be at least 3 users executing this script. The time interval between the initiation of a LOGIN command and the completion of the LOGOUT command by each user shall not exceed 130 seconds.

C. <u>Script 3</u>

<u>Commands</u>

1. "LOGON" with password "FACT,NSRDC".
2. Enter the "Program Development" subsystem.
3. Perform functions as required to initiate the creation of a new "text file".
4. Build the "text file" by entering the FORTRAN language statements contained in the file "BENCH 1".
5. Perform the following editing functions:
   - search the "text file" for occurrences of each arithmetic operator (4,**)
   - search, display, and change all occurrences of the character strings "F(" and "F=" to "G(" and "G=" respectively.
6. Execute a FORTRAN compilation of "BENCH 1".
7. "SAVE" the resultant binary file.
8. Load and execute the binary file 5 times.

Table II. Background Job Stream (continued)

9. "QUERY" the system to obtain the file names of files currently "ATTACHED" to this job.

10. "PURGE" all files "SAVED" during execution of this job.

11. "LOGOUT".

12. Repeat 1 thru 12 as required for continuous execution during the benchmark test.

There shall be at least 34 users executing this script. The time interval between the initiation of a LOGIN command and the completion of the LOGOUT command by each user shall not exceed 534 seconds.

D. <u>Script 4</u>

<u>Commands</u>

1. "LOGON" with password "MED,NSRDC".

2. "ATTACH" file whose file name is "BENCH 2".

3. Execute a FORTRAN compilation of "BENCH 2".

4. "SAVE" the resultant binary file.

5. Load and execute the binary file 5 times.

6. "PURGE" all files "SAVED" during the execution of this job.

7. "LOGOUT".

8. Repeat 1 thru 7 as required for continuous execution during the benchmark test.

There shall be at least 4 users executing this script. The time interval between the initiation of a LOGIN command and the completion of LOGOUT command by each user shall not exceed 238 seconds.

E. <u>Script 5</u>

<u>Commands</u>

1. "LOGON" with password "SLOW,NSRDC".

2. "ATTACH" file whose file name is "BENCH 3".

3. Execute a FORTRAN compilation of "BENCH 3".

4. "SAVE" the resultant binary file.

5. Load and execute the binary file 2 times.

6. "PURGE" all files "SAVED" during the execution of this job.

7. "LOGOUT".

8. Repeat 1 thru 7 as required for continuous execution during the benchmark test.

There shall be at least 2 users executing this script. The time interval between the initiation of a command and the completion of the LOGOUT command by each user shall not exceed 238 seconds.

F. <u>Script 6</u>

<u>Commands</u>

1. "LOGON" with password "DELAY,NSRDC".

2. "ATTACH" file whose file name is "BENCH 4".

3. Execute a FORTRAN compilation of "BENCH 4".

4. "SAVE" the resultant binary file.

5. Load and execute the binary file.

6. "PURGE" all files "SAVED" during the execution of this job.

7. "LOGOUT".

8. Repeat 1 thru 7 as required for continuous execution during the benchmark test.

There shall be at least 1 user executing this script. The time interval between the initiation of a command and the completion of the LOGOUT command by each user shall not exceed 248 seconds.

Approximately 15% of the logged-on users issued quick response service requests.

e. Medium Response. Less than 20 seconds response for loading and executing a medium sized user program. The CPU and central memory requirements (expressed in CDC 6600 units) for executing a medium sized user program are projected as normal distributions with means of 1 CPU seconds and 40K (octal) central memory words.

Approximately 5% of the logged-on users issued medium response service requests.

f. Slow Response. Less than 60 seconds response for loading and executing a large user program. The CPU and central memory requirements (expressed in CDC 6600 units) for executing a large user program are projected as normal distributions with means of 3 CPU seconds and 40K (octal) central memory words.

Approximately 5% of the logged-on users issued slow response service requests.

g. Delayed Response. Less than 120 seconds response for loading and executing a CPU bound user program. The CPU and memory requirements (expressed in CDC 6600 units) for executing a CPU bound user program are projected as normal distributions with means of 9 CPU seconds and 40K (octal) central memory words.

Approximately 2% of the logged-on users issued delayed response service requests.

The job streams were described as a series of conversational scripts which specified the set of commands which were to be executed.

Script items were described in functional terms to avoid terminology peculiar to any single vendor's command language. It was necessary for the vendor to translate script commands into the command language provided as part of the proposed system.

The background scripts, listed in Table II, were to be translated and executed in the following manner. Multiple copies of scripts 1 through 5 and a single copy of script 6 were to be executed concurrently. The number of copies of each script was based on the mix described in the above specifications. The vendor was to begin the execution of the 49 background scripts using a Remote Terminal Emulator (RTE) [8]. Timing considerations included the execution time for each script (as specified in Table I), a think time of 12 seconds per command (based on observed user characteristics), and a data transfer time of 30 characters per second (based on the capabilities of the RTE's).

When all 49 background jobs were being executed, the vendor was to begin executing the one foreground script (described in Table III) via an interactive keyboard/printer terminal located at the benchmark facility. Since the foreground script was manually executed by vendor personnel, no requirements were established concerning the think time or typing rate. At the completion of the test the vendor's standard accounting software was used to collect, analyze and report the load imposed by each background job stream, in terms of CPU, central memory and I/O resources utilized.

In all, 50 jobs the commands were to be entered one-at-a-time in a strict sequential manner such that the response from the most recent command would be received before another command would be entered, i.e., no type-ahead was permitted. The response time associated with the execution of each command in the foreground job was electronically recorded and measured using the Network Measurement System (NMS) developed by the Institute for Computer Sciences and Technology of the National Bureau of Standards [1].

3.0 THE NBS NETWORK MEASUREMENT SYSTEM

The NBS Network Measurement System (NMS) was developed to measure interactive computer networks, teleprocessing systems, and network services by focusing on the service delivered to users rather than on the internal operating efficiency of the system. The tedium of attempting to collect and analyze the same data by manual means (when possible) is so error prone that without such a tool, some other evaluation approach would have to be employed. The information obtained aids users in the quantitative evaluation of such systems and services. The Network Measurement System consists of a data acquisition system and a separate set of data analysis programs.

The data acquisition system, called the Network Measurement Machine (NMM), is implemented on a DEC PDP 11/20 which features a high precision programmable clock and a collection of communications

Table III. Benchmark Job Stream

1. "LOGON" with password "SAME,NSRDC". Response time for procedure required to log-on and validate a user for access to the system will be tested.

2. Enter "ANTIDISESTABLISHMENTARIANISM". Character echo time will be tested if system is operating in full duplex mode.

3. Perform the "CHARACTER BACKSPACE" function 28 times. Character backspace time will be tested.

4. Enter text "The Quick Brown Fox Jumped Over the Lazy Dog's Back 1234567890 Times" and delete. Line delete time will be tested.

5. "ATTACH" file and file name "BENCH 1". Response time for the "ATTACH" function will be tested.

6. Perform function to "LIST" file at user terminal and then perform the "OUTPUT ABORT" function while file is being listed. Response time for the output abort function will be tested.

7. "ATTACH" the files with file names "BENCH 2", "BENCH 3", and "BENCH 4".

8. "QUERY" the system to obtain the file names of the files currently "ATTACHED" to this job. Response time for the query function will be tested.

9. Execute FORTRAN compilations for "BENCH 1", "BENCH 2", "BENCH 3", and "BENCH 4". "SAVE" the resultant binary files.

10. "COPY" the binary files resulting from compilations of "BENCH 1" and "BENCH 4".

11. "SAVE" the file copied from the binary version of "BENCH 4". Response time for the save function will be tested.

12. "PURGE" the file "SAVED" in step 11. Response time for the purge function will be tested.

13. Enter the "Program Development" subsystem. Time for accessing a subsystem will be tested.

14. Create the control card records necessary to submit the file copied from the relocatable binary version of "BENCH 1" for execution as a batch job. Make a deliberate error in one of the lines of text. Use any intra-line editing command except character backspace and line delete to correct the typing error.

15. Exit from the "Program Development" subsystem "SAVING" files as necessary.

16. Submit the control card records created in Step 14 for execution as a batch job. Response time for acknowledgment of submission of a batch job will be tested.

17. "QUERY" the system for the status of the batch job submitted in Step 16. Response time for query will be tested.

18. Load and execute the binary version of "BENCH 1". Response time will be tested.

19. Load and execute the binary version of "BENCH 2". Response time will be tested.

20. Load and execute the binary version of "BENCH 3". Response time will be tested.

21. Load and execute the binary version of "BENCH 4". Response time will be tested.

22. Load and execute the binary version of "BENCH 4" and then perform a "TASK ABORT" while the program is executing. Response time for task abort function will be tested.

23. "LOGOUT". Logout time will be tested.

interfaces [6]. In addition, special hardware is provided to connect the NMM to the network that is to be measured. This special hardware includes portions of a general purpose intercommunications system as well as an automatic calling unit and line selector system for computer-controlled originating and answering of data calls via common carrier communications facilities. The basic function of the NMM is to receive characters from both computer and user and to record these characters along with information as to the source of the character, the time at which the character occurred, and the state of the communication line at the time the character existed. Aside from this basic function, the most important feature of the NMM is that it does not perturb the network being measured.

3.1 Pre-Benchmark Test Utilization of the Network Measurement System

DTNSRDC first verified the Network Measurement System to their satisfaction. The verification consisted of dialing through the Network Measurement Machine to two different Government owned timesharing systems. DTNSRDC connected a strip chart recorder to their terminal so that they could compare the data recorded in this manner with the data recorded by the NMM. NBS supplied DTNSRDC with a formatted dump of the data recording tape for this purpose. The strip chart was, as expected, extremely inconvenient to use even for this verification. It would have been prohibitively difficult to use as the data acquisition mechanism in the live test demonstration.

Each vendor that responded to the solicitation was given several opportunities to schedule test demonstrations of the NMS. They were provided with simple instructions for dialing the NMM and instructing it to in turn dial their computer. The purpose of making the NMS available for a test demonstration was to acquaint them with the information that would be provided to DTNSRDC. There were two types of output. One was a formatted dump of the data tapes collected by the NMM (referred to above), and the second was the statistical analysis of this recorded data as produced by this data analysis program [7]. It is believed that the vendors performed their own timing, which they compared with the NMS output.

3.2 Testing The Correct Operation of the Network Measurement Machine

The utilization of the NMM in a procurement effort requires that precise procedures for its testing and operation be set forth. The general areas addressed are: activity logs, equipment validation, and software validation.

Activity logs were of several kinds: tape log, NMM console log, and operator's log. The tape log specifies the tape format, character set, density, etc. It identifies the files on the tape and their contents.

The NMM console log is simply the NMM operator's console output. For procurement efforts, however, it was considered to be necessary to use a special version of the NMM which was generated to support a hard copy terminal as the operator's console.

The operator's log is on the order of a diary, tracing all activities which transpire during the entire session. These activities include testing of NMM hardware and software, including hardware diagnostics if deemed necessary. Also included in the operator's log is a record of any anomalies which may have occurred during the session, as well as steps taken to deal with these situations.

The criteria applied to insuring the integrity of the NMM is that of comparing the results of the recorded data with that of a known input. A favorable comparison is in itself sufficient to insure that the timing mechanism in the NMM is working properly, and that other functions (i.e., recording the data on magnetic tape, etc.) are being performed.

To achieve this comparison, the NMM is installed in its normal configuration. A character generator connected to the NMM sends characters of a known rate and known inter-character interval. The NMM, if functioning properly, will record these characters. A comparison between the known rates and the recorded rates will reveal the operability and accuracy of the measurement machine.

The application of the procedure used to test the NMM requires a specially modified ASCII code generator. This generator uses a crystal clock for high stability. A special inter-character rate selector switch was added. This allows the operator to select inter-character times of 0.1, 1.0, and 10 seconds. The generator is connected to the NMM, with its switches set to allow the timer to trigger the character generator which is set to increment through the ASCII character set.

Once the data is collected, a special program is invoked which prints the contents of the data tape. This program produces a listing of each character received by the NMM and the inter-character time differences. Observation will quickly indicate if the NMM is functioning properly. One can quickly scan the difference column and compare it to the generator setting. Also, each character can be quickly checked for correct temporal sequence as the generator produces a monotonically increasing sequence of ASCII characters.

This testing system is designed to convince the NMM operator that the machine is operating correctly before an experiment is actually run. Other uses can be made of the timing device while the NMM is running. The operator can monitor the generator port as the generator sends data. If the data from the monitored port is inconsistant, the operator can assume that the code generator or the NMM is malfunctioning.

The analysis routines can also make use of the data generated by this timing device. They can use the data from the generator port to verify that the NMM behaved properly for the duration of the experiment by comparing the inter-character differences. If the analysis routines detect any discrepancies, the entire data collection experiment should be suspect.

4.0 BENCHMARK TEST ON A VENDOR'S SYSTEM

This section completes the description of the procurement selection by presenting descriptive data on the vendor system for which award was made.

4.1 Benchmark Test - Burroughs Corp. B7700 System

The Burroughs Corp. began the Benchmark test by cycling execution of the background job streams described in Table II. Sixteen minutes later the benchmark

Table IV. Burroughs Benchmark Background Script Execution

SCRIPT NUMBER	SCRIPT CLASS	REQ'D TIME	MIN. TIME	MAX. TIME	AVE. TIME
1	1	6.02	5.10	5.30	5.17
2	4	3.97	1.30	2.80	2.20
3	3	8.90	5.89	6.98	6.43
4	3	8.90	6.09	6.94	6.42
5	3	8.90	5.97	6.73	6.31
6	3	8.90	6.11	6.65	6.34
7	3	8.90	5.97	6.37	6.22
8	2	2.16	0.52	0.67	0.57
9	1	6.02	5.07	5.24	5.15
10	4	3.97	1.28	2.74	2.19
11	3	8.90	6.16	6.66	6.39
12	3	8.90	6.03	6.55	6.35
13	3	8.90	5.96	6.68	6.45
14	3	8.90	6.26	6.57	6.41
15	3	8.90	6.14	6.38	6.29
16	2	2.16	0.51	0.71	0.58
17	1	6.02	5.06	5.28	5.16
18	4	3.97	1.00	3.16	2.13
19	3	8.90	6.23	6.77	6.57
20	3	8.90	6.11	6.69	6.43
21	3	8.90	6.19	7.21	6.46
22	3	8.90	6.17	7.02	6.41
23	3	8.90	6.17	6.76	6.38
24	3	8.90	6.10	6.62	6.27
25	1	6.02	5.05	5.32	5.16
26	4	3.97	1.63	2.45	2.05
27	3	8.90	6.19	7.05	6.53
28	3	8.90	6.41	6.76	6.47
29	3	8.90	6.14	6.70	6.43
30	3	8.90	5.99	7.20	6.60
31	3	8.90	6.34	6.77	6.57
32	3	8.90	6.21	6.75	6.51
33	1	6.02	5.05	5.28	5.14
34	5	3.97	1.29	1.94	1.67
35	3	8.90	6.26	7.55	6.68
36	3	8.90	6.18	6.96	6.57
37	3	8.90	6.03	6.99	6.54
38	3	8.90	6.35	7.30	6.60
39	3	8.90	6.21	7.10	6.66
40	3	8.90	6.11	6.89	6.48
41	6	4.13	1.14	2.26	1.66
42	5	3.97	1.28	2.10	1.61
43	3	8.90	6.25	7.37	6.75
44	3	8.90	6.35	7.02	6.63
45	3	8.90	6.61	7.02	6.81
46	3	8.90	6.34	7.01	6.63
47	3	8.90	6.41	6.81	6.67
48	3	8.90	6.22	6.60	6.48
49	2	2.16	0.51	0.71	0.66

window was established by initiating and repeating ten (10) times the foreground (Benchmark) job stream described in Table III. (The vendors were permitted to repeat the foreground job stream ten times and use the average execution time as the accepted time.) Table IV and V summarize the accepted background and foreground execution times. Table IV indicates that all background scripts ran in less than the maximum permitted time (labeled REQ'D TIME) which was obtained by adding the component times for think time, transfer rate, and response time for all the commands in the script. Table V also indicates that nineteen of the twenty foreground functions ran in less than the maximum accepted time. The vendor could have failed two of the twenty functions and have been acceptable.

Although not employed as part of the selection process, the vendor was permitted to run an internal driver to produce the background load. Of course, the internal driver was not system resident when the RTE was used. It is interesting to compare the effect of using the internal driver as compared to the RTE on the timing of the foreground job. The average times obtained using the internal driver are presented as the last column in Table V.

Table V. Burroughs Benchmark - Foreground Functions

FUNCTION NR.	FUNCTION DESCR.	MAX.ACC. TIME	MIN. TIME	MAX. TIME	AVE. TIME	INTR. DRIVER AVE.
1	Log-on	6.0	.74	1.65	1.15	1.6
2	EOM to CR	0.2	.05	.05	.05	.05
3	Char. Echo	0.2	.05	.05	.05	.05
4	Char. Backsp	0.2	.12	.24*	.16	.16
5	Line Del	0.2	.05	.05	.05	.05
6	File Attach	3.0	.70	3.11*	1.57	1.74
7	Output Abort	1.0	.70	2.24**	1.21***	1.20
8	File Status?	3.0	.80	1.46	1.13	1.54
9	File Save	3.0	.97	2.99	1.64	1.93
10	File Kill	3.0	.34	1.66	1.05	2.17
11	Call Utility	3.0	.50	1.01	.75	1.17
12	Edit Line	3.0	.32	1.44	.63	.47
13	Batch LGO	3.0	.64	1.96	1.11	1.51
14	Job Status?	1.0	.32	2.40*	.59	1.04
15	LGO pgm sm.	6.0	1.87	11.51*	3.50	5.14
16	LGO pgm med	20.0	2.49	11.86	5.52	8.92
17	LGO pgm lg	60.0	4.31	12.34	8.37	17.80
18	LGO pgm cpu	120.0	14.04	21.51	19.08	28.16
19	Task Abort	1.0	.30	.99	.41	.49
20	Log-out	6.0	.31	.98	.50	.45

*-A single out-of-spec recording in 10 runs
**-Four out-of-spec recordings in 10 runs
***-The vendor was required to pass 90% of the functions

5.0 CONCLUSIONS

The work reported in this paper constitutes the first application of the NBS Network Measurement System in a procurement selection exercise. In addition to the careful attention to detail which must be part of any such exercise, this first application was evaluated to assess the applicability of both the NMS and the methodology which it embodies as a production procurement selection tool. No difficulties were encountered. The measurement of the service delivered to the user at the terminal proved valid. The response time data would have been impossible to collect without an automated instrument. Manual transcription and data analysis would have been prohibitively expensive and error-prone. The NMS has proven to be a valid and useful implementation of this methodology.

The NMS prototype can be re-implemented in hardware less expensive and more portable than the shared resources which were employed. Following the application described herein, NBS has developed specifications for, and initiated procurement of, a self-contained single user terminal NMS, complete with report generation capabilities.

Use of the NBS system permitted a quantitative analysis to be made of the "quality of service" the selected system would provide the DTNSRDC user community.

This paper incorporates parts of an unpublished report, "Computer Performance Evaluation at the David Taylor Naval Ship R&D Center," prepared by A. Hankinson, K. Rieck, T. Small, B. Tinker, and D. Wilshe. The cooperation of the U. S. Navy ADP Selection Office (ADPSO) and Burroughs Corporation is also acknowledged.

BIBLIOGRAPHY

[1] Abrams, M. D., Cotton, I. W., Watkins, S. W., Rosenthal, R., and Rippy, D. E., "The NBS Network Measurement System," _IEEE Transactions on Communications_, October 1977, pp. 1189 - 1198.

[2] Abrams, M.D., and Cotton, I.W., _The Service Concept Applied to Computer Networks_, NBS Technical Note 880, August 1975.

[3] _Guidelines for Benchmarking ADP Systems in the Competitive Procurement Environment_, Federal Information Processing Standards Publication 42-1, May 1977.

[4] _Guidelines for the Measurement of Interactive Computer Service Response Time and Turnaround Time_, Federal Information Processing Standards Publication 57, August 1978.

[5] Miller, R. B., "Response Time in Man-Computer Conversational Transactions," _Proc. Fall Joint Computer Conference_, 1968, pp. 267-277.

[6] Rosenthal, R., Rippy, D. E. and Wood, H., _The Network Measurement Machine -- A Data Collection Device for Measuring the Performance and Utilization of Computer Networks_, NBS Technical Note 912, April 1976.

[7] Watkins, S.W., and Abrams, M.D., _Interpretation of Data in the Network Measurement System_, NBS Technical Note 897, February 1976.

[8] Watkins, S. W., and Abrams, M. D., _Survey of Remote Terminal Emulators_, NBS Special Publication 500-4, April 1977.

BENCHMARKING IN SELECTION OF TIMESHARING SYSTEMS

D.J.M. Davies

Department of Computer Science
The University of Western Ontario
London, Canada N6A 5B9

This paper discusses the role of benchmark performance evaluation tests in selecting systems for timesharing and general purpose computation. Since performance evaluations can be expensive for both customer and vendor, and since they provide only limited information, a methodology is required for making them economical as well as effective. This is discussed with respect to the evaluations performed during a recent computer selection process at The University of Western Ontario.

The U.W.O. benchmarks were intended to be reasonably economical to run. Their structure and design are discussed, and their successes and failures are analysed. Some of the problems were specific to particular machines, others were more generalized. Some sample results are presented to illustrate the discussion.

Key words: Benchmarking; Computer System Selection; Response Times; Timesharing Systems.

1. Introduction

The University of Western Ontario has recently completed the selection process to choose a successor for the old DECsystem-10/50. The final decision was, in fact, to purchase a new, larger DECsystem-10.

The selection process was conducted mainly by a selection committee, operating under the authority of the University Computer Council. The committee included representatives from the user community and from the Computing Centre. It was decided at an early stage first to draw up a list of requirements, then to issue a Request for Proposals (RFP), next to reduce the proposals received (if any) to a short-list of strongest candidates, and finally to recommend a final choice after detailed evaluation.

Benchmarking was involved in the last two steps of this. The RFP required the vendor to run a very small set of programs, to enable the committee to get an impression of overall differences in system performance. Then a more elaborate set of programs was run under supervision on each of the short-listed systems.

The benchmarks were not intended to be overwhelmingly important in comparing different systems. The systems were evaluated according to criteria outlined in the RFP, under the headings of Price, Performance, Flexibility, Expandability and Capacity, Compatibility with existing system, and Availability of system support.

2. Benchmark Design Considerations

Performance tests cannot, by themselves, indicate what choice to make between competing general-purpose time-sharing systems. There are many other considerations, such as price, quality of system support (hardware and software), range of software capabilities, quality and range of language implementations, documentation, ability to meet special requirements such as in communications equipment, and so on.

The workload in a general purpose system is usually very variable, and future workloads and requirements cannot ordinarily be predicted in detail. They will depend to some extent on what machine is selected, and what that machine facilitates. Measurements have been made on several systems, to characterise (at least in part) the behaviour and demands of users [1, 2, 3].

It is apparently not uncommon for customers (particularly government agencies) to request vendors to create and run performance tests to given specifications. Such tests can sometimes cost a vendor $100,000 or more, and involve much duplicate work by those who tender.

Naturally, such tests are regarded by the vendors as wasteful, particularly of their more skilful applications programmers who are diverted from other customer support services.

The U.W.O. Selection Committee decided that some performance tests would be necessary, because it is difficult to describe an expected time-sharing workload unambiguously. Tests would ensure that systems proposed by vendors were compared in a fairly objective manner instead of having to trust all the salesmen to understand our requirements in the same way.

In effect, the benchmark program mix became a *de facto* definition of our expected workload. This was so because we provided the programs and specified how they were to be run (and monitored any changes made for specific machines). Originally, however, we prepared the programs partly to ensure uniformity between vendors, and partly to minimize total costs and time by reducing repetitions of program development.

The Benchmarks were to be used (i) to judge how capable a system was of serving 60 to 90 time-sharing users, (ii) to judge how capable a system was of being expanded to handle 120 or 150 users simultaneously and what this would require in hardware terms, and (iii) to aid in determining exactly what initial hardware configuration should be considered for purchase. They also provided an opportunity for the selection committee to get some exposure to the various operating systems and language implementations, by observing exactly how the original programs had to be modified to run on them, and by using the systems directly during benchmark runs.

3. Initial RFP Benchmarks

The RFP required running of three tests, and invited running of a fourth.

Test Number 1 was a Fortran program that executes a series of iterative loops and times the speeds of certain basic computations. The program measures the times elapsed and CPU charged internally, and prints the results. Subtests measured the times taken for various standard functions, fixed point operations, calling a null subroutine, and single and double precision arithmetic.

This program was to run in an otherwise empty computer. Figure 1 shows how some of the various systems differed. The iteration counts were set so that all subtests took equally long in the DEC-10/50. An interesting feature is that some systems were disproportionately slow in calling a (null) subroutine. This could influence users' programming style. Some important types of computation, such as character string manipulation, were not tested, and overall the test does not necessarily predict relative performances in *real* programs.

Test Number 2 was a Fortran program that writes and reads disk files in unformatted transfers, of one disk block, sector or PRU at a time. Some of the files were sequential, and some were written and read randomly. Test Number 3 was a Cobol program that writes and reads files on magnetic tapes, using record lengths varying from 64 to 2040 bytes, and at 1600 and 800 bpi. Three drives were involved in all, with disk-to-tape copies, and comparisons between disk and tape.

The programs for tests 2 and 3 similarly measured elapsed and CPU time and were to be run on an otherwise empty system. The program output permitted calculation, for each subtest, of how many records were transferred per unit of CPU and of elapsed time. The results showed fairly clear trends which permitted the system to be ranked in terms of stand-alone I/O throughput capability. (However, the performance was sometimes different during the later supervised benchmark.)

The first test permitted us to ensure at least that the proposed systems had somewhat more processor power than the old

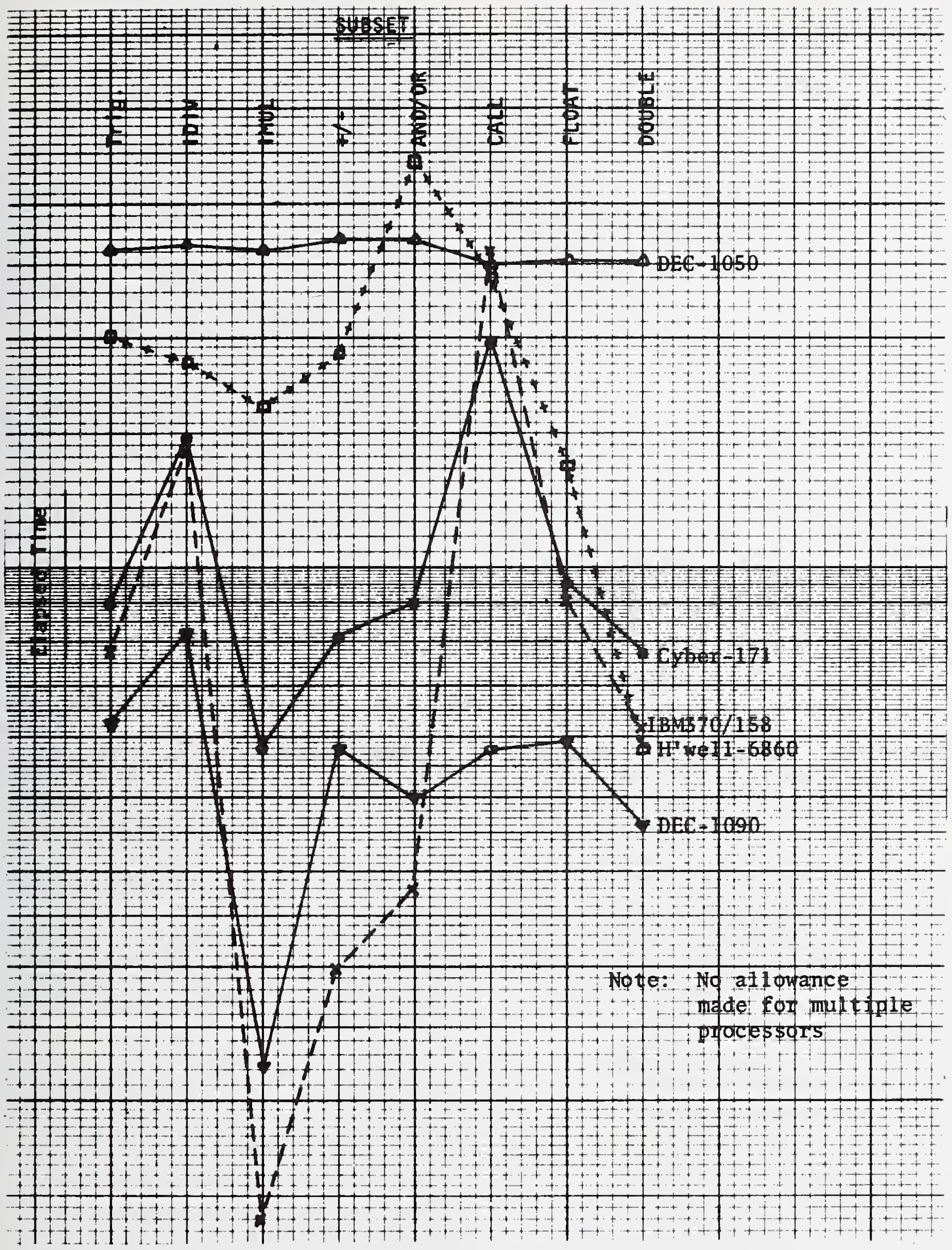


Figure 1. Comparative Results of Test Number 1 .

DEC-10/50. The second and third tests checked that the systems could perform I/O fairly quickly. It was to be left to the final detailed benchmark to determine how much of this potential 'stand-alone' power was actually delivered to users in heavily loaded time-sharing.

Test Number 4 was a simple synthetic benchmark to test the ability of the system to handle many jobs simultaneously. It was not required to be run; a vendor could instead report the results of an analytical calculation or simulation. About half of the proposals received included results from an actual run.

The test comprised running totals of 30, 45, 60, 90 and 120 user jobs or processes simultaneously, using four classes of jobs in a completely synthetic workload. The jobs were Edit-type, I/O-bound, Small compute, and Large compute, all created from a common Fortran program initialised with one of four block data files respectively.

The programs all read an input line (of 10 characters, whose content is ignored), perform some computation and possibly some disk I/O, print an output of 40 characters, and wait for the next input. The stimulator was supposed to take 16 seconds to type the 10 characters of input. This rate of typing is based on data reported from Leeds [4] indicating that users typically type at about 1 second per character, rising to 1.9 seconds per character for more elaborate work, so we averaged at 1.6 seconds per character. This is consistent also with a report [2] that input lines average 10-12 characters and responses about 38-46 characters mean line length. Another similar benchmark [5] required input to be typed at a mean rate of 1.1 characters per sedond, which allows a little less thinking time than ours.

This fourth test required the mean response times to be quoted for the edit-type jobs, which composed 3/4 of the jobs. The way the jobs were set up gave all vendors some difficulty. First, the programs were "normalised" for processor speed, with the edit-type programs using 10 msec. of CPU time per interaction irrespective of processor power. Secondly, the programs 'touched' all of their address space in each interaction, by scanning the memory (mostly filled by an array). This latter feature was no problem for swapping type systems, but created untypically heavy paging rates on VM (demand paging) systems. This was aggravated by the lack of 'random' variations in the activities of the program. A tendency developed on some systems towards synchronisation of different processes with each other through side effects of the page turning mechanism.

The object of this test was to examine the response times as a function of system load, to determine at what point the responses started to deteriorate significantly. We hoped the 'elbow' would be at not much less than 90 users, preferably somewhat higher. However, with the VM/370 system in particular, we observed only the consequence of heavy paging, which apparently saturated the swapping channels at a point well below full processor utilization.

In the end little significance was attached to the results of this last test, but it warned us of some pitfalls to avoid in the later 'big' benchmark.

The first three tests were designed to finish within about 5 to 10 minutes each, and should have been easy and economical to perform. The results were not hard to interpret for trends in relative power, though individual subtests are not meaningful in isolation. The fourth test was expected to be harder to set up, so it was not mandatory to run it. What results we did obtain from it were also of questionable value.

4. The Major Benchmark

After examination of the proposals, a short-list was developed of systems judged worthy[1] of a more detailed assessment. These systems were the CDC Cyber 171, the DEC System-10/90 and Honeywell Level 66 and Multics system.

Later, when a more substantial time-sharing benchmark work-load had been developed, the short-listed vendors discovered that they had underestimated our computing requirements, and all these systems had to be enlarged beyond the original proposals. The Cyber-171 was upgraded to a 172 or 173, the Level 66 proposal was dropped, and the DEC System and Multics were expanded.

The benchmark was again a mixture of 'edit-type', I/O, and CPU-bound tasks, but 'real' programs were used instead of totally artificial programs. The scripts were artificial still, but used the real

[1]Some good systems were turned down, because of price or other factors.

editor and real language systems (mainly BASIC and FORTRAN). Actually, modified versions of the programs for the first three tests were included, as CPU and I/O bound scripts, alongside other scripts as well. We used three different 'edit-type' scripts, and five I/O-bound and four CPU-bound scripts. The system SORT was used, and an APL script, and a Fortran program from one committee member's research project, among other tasks.

Again, we looked mainly at the response times received on editing-type scripts, but also examined other scripts to ensure they were not completely stalled. We examined the behaviour as a function of number of total jobs (40, 60, 90 and 120) and with several different machine configurations (depending on time available and the vendor).

The terminal transactions for all scripts and runs were brought back, on several magnetic tapes for each machine.

It was intended originally to analyse the scripts with Fortran programs on the U.W.O. Cyber computer, but in the end I analysed most of the data from the DEC-10/90 and Multics using the POP-10 system [6]. This permitted me to develop interactively different approaches towards analysing the scripts. The output from CDC runs was not amendable to processing in POP-10, and was analysed afterwards on the Cyber in a more limited way.

Apart from running standard job mixes at various levels of usage, we also attempted to explore the effect of mistaken assumptions about the future workload. Therefore, at the 60 user level, we also ran "Margin" tests with (i) more I/O, (ii) more CPU, and (iii) more CPU and more I/O scripts, with fewer editing scripts.

5. Results Obtained

In the end, a great deal of useful data was obtained from the benchmarks, but not so much as we hoped. There were various factors that reduced the amoung of information extracted.

5.1 Timestamps

Because most of the programs did not print any timing information of their own, we had to rely on 'timestamps' recorded by the respective stimulators. These varied in nature, and also were sometimes hard to interpret - we were not always sure exactly what the figures referred to. This also meant the data analyses had to be performed quite separately for the systems, creating extra work. More internal metering by programs would have aided interpretation of data on overall consumption of time.

5.2 Stimulator Output

Nevertheless, internal metering in the programs obviously cannot determine what response times are observed at the actual terminal. The DEC system used an external stimulator while Honeywell and CDC used internal stimulators.

The output from the DEC system was very difficult to interpret, particularly in the scripts for the editor TECO. This was mainly because the stimulator recorded times on a line by line basis, but TECO commands are terminated by an escape instead of a carriage return. We were not sure exactly at what time an input to TECO was terminated. In addition the stimulator appeared sometimes not to wait for the end of a response before starting the next input, though the interpretation of the scripts remains unclear. However, the DEC output showed clearly how long a job took to complete one cycle of its script.

The Multics system (Honeywell) did not provide absolute timestamps, but only incremental elapsed time information. Response times were easily extracted, but the time to complete a whole cycle had to be obtained by adding up the various increments, and it turned out that in certain circumstances, there would be elapsed time not accounted for. The most important source of lost time was any response involving more than one line of output, with computation delays between the first and last output lines.

5.3 Identifying Scripts

The CDC Cyber system did enable response times and cycle times to be measured, but the outputs from different scripts were all jumbled up almost indiscriminately so a great deal of computation was required to find all instances of any one script. This was aggravated by lack of a clear identification at the start of each script cycle.

5.4 Changes In Load

Our benchmark specified running with different numbers of jobs. In some runs on Multics and most runs on the Cyber, the number of jobs was changed in mid-stream during a single run. This was indeed the way it had been planned. However, when it came to analysing the data, this made it much harder to tell how many jobs had been running when a particular piece of script output was generated. So it was complicated to determine how response and cycle times depended on the workload. Any one run should be at a fixed load level, even if this does take extra time in starting and stopping the system.

5.5 Overloads

In the Request For Proposals, it had been indicated that the response times seen by editing-type jobs (highly interactive ones) would be the most important feature of the performance. However, it was seen as important that the less interactive jobs should also procede at a moderate pace. When the systems were actually loaded up, they differed considerably in how they reacted.

One of the systems appeared in effect to freeze some of the jobs (particularly CPU-bound ones) and devote attention to interactive jobs (and even then, selectively). At 90 and 120 users, apparently identical jobs would proceed at very different rates. The system was, of course, being forced into overload. In fact, we obtained no useful data with 120 jobs, because some of them simply did not get going.

Another system performed more smoothly under heavy overload, in that it kept all jobs moving even under the 120 user load. The price it exacted was rather slower response times to interactive jobs, particularly when file I/O was involved. Both of these systems appeared to have a disk I/O bottleneck at 60 users or above, but it manifested itself very differently in the two cases.

The Cyber system showed the curious behaviour of worse response times if the central memory was enlarged beyond 128K words. This appeared to be an artifact of our workload and the scheduler, such that the scheduler tended to hang on to compute-bound jobs in central memory to keep the CPU always busy. When central memory was increased, this gave the interactive jobs more competition and they got service less often in our particular mix.

5.6 Synchronisation

It was mentioned earlier that some systems suffered from synchronisation effects in the initial Test Number 4. The scripts for the major benchmark again showed clear evidence of this in some cases. For example, it turned out during analysis that some TECO editor scripts had proceeded in a kind of lock-step. (There were 23 of them on the same script at 60 user level). This seems to have resulted because all scripts started off simultaneously, met long delays when all simultaneously requested file I/O, and all ran rapidly in between I/O activities.

This happened even though we had specified some random variations in typing speed ("thinking time"). As it was, we had only 12 different scripts in use, and even starting off different scripts with random delays might not have been much help. It would appear that no single script should be followed by more than 10-15% of the jobs. This would, of course, increase further the labour of creating and analysing the benchmark.

5.7 Typing Speed

Real typists do not type at a constant speed of 1.1 characters per second or whatever. Measurements of typists' speeds typically show a skew distribution, with some very quick inputs, and some taking three or more times the mean [1, 3]. The vendors' stimulators did provide a capability to randomly vary the think times, but apparently only in uniform distributions which produced neither very short nor very long delays. This possibly aggravated the effect of having many jobs following the same script.

The think times are critical to the performance of the whole benchmark - if they are too short, the system throughput will not increase beyond its capacity, and the response times perceived will merely get longer. Indeed, it was not ultimately clear whether response times are all that meaningful in isolation, and we also examined the times required for jobs to complete a cycle of the script. We did drive all systems tested to their capacity, observing the effects of doing so.

5.8 Software Environment

The point was made by the vendors, and has also been made elsewhere [7] that benchmarking in this way completely ignored the effects of the software environment. Typically, a high fraction of all computer time used goes on a variety of activities which are known as software development and maintenance [8,9]. The time spent by users at a terminal or on other programming activities will depend very much on the quality of the software tools provided. If a vendor has an excellent diagnostic compiler which enables the user to develop his program with only 60% the number of attempts, then this can more than compensate for a slightly slower machine.

In this vein, the quality of the standard software is (or should be) of crucial importance. Time lost coping with problems in the standard software is time completely wasted. It is well established that the use of a high-level language for systems programming can considerably reduce its cost and increase the reliability of the product [9, 10]. Virtually all major vendors are now moving in this direction.

6. Benchmark Evaluation

It was very difficult to know how to compare the different performances of the systems, given these considerations. All vendors claimed (doubtless with justice) that performance could be tuned in any way that we wanted; we had asked them not to tune the system specially. We did learn that in each case a more powerful disk subsystem and more memory were needed to handle 60 users well than had originally been expected.

In the final evaluation, the performances of editing type jobs and the I/O and CPU jobs were computed separately, and then combined, with maximum weight given to the editing-type jobs.

The results of the margin tests were not used, because there was insufficient data to be confident about any trends that might have appeared. The responses were worse, of course, for more intensive workloads.

In comparing response times from different computers, the ranking depends on what criterion is used. Figure 2(a) shows the frequency distributions of various response times in a script using the appropriate editor, comparing the DECsystem-10 and the Multics system, at 60 users in our standard mix.

The response time distribution in Multics was unimodal and not very skew. The mean and median were close together at about 2.1 seconds for 60 users. Few responses were fast (under 0.5 seconds) but none exceeded 7 seconds at 60 users in this particular run. The DECsystem-10 responses were much more varied, with a multimodal, very skew distribution. The principal peak was also the largest, and the median response time was 0.14 seconds. However, responses larger than that tended to be very long, with the longest response being 58 seconds. Detailed study showed that the quick responses were for commands not involving disk I/O and the slow ones involved disk I/O. The mean response time of 5.2 seconds was very different from the median. 10% of the responses exceeded 12 seconds, and 20% exceeded 10 seconds.

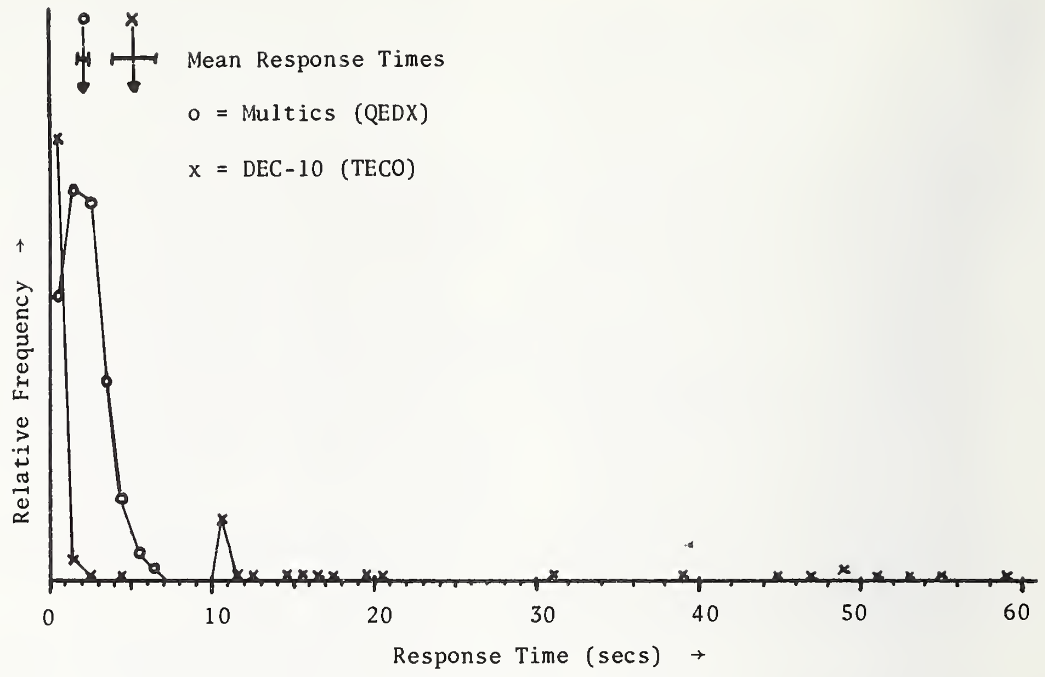
Given this performance, it is hardly representative to use the medians for comparisons, because this shows nothing of the spread in DEC-10 responses. The overall means were therefore used for scoring.

On this particular script, the Multics system definitely performed better than the DECsystem-10/90. On other scripts, however, the differences were reduced or reversed as in Figure 2(b), and as indicated earlier, the DECsystem-10 performed better than Multics on the CPU-bound scripts in our particular job mix. The earlier Test Number 1 had shown the DEC CPU to be quite fast; the Multics system was suffering from a heavy burden of page faults, due to too small a primary memory.

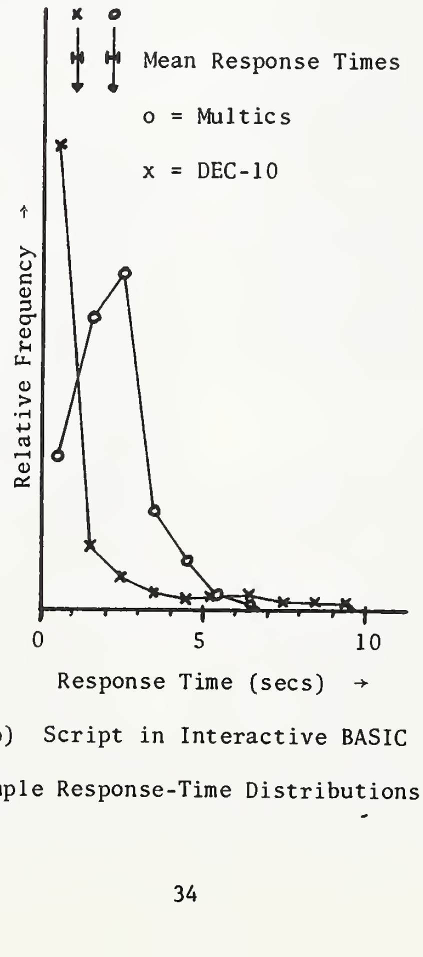
To compare the mean response times (or mean cycle times) from different machines and runs, it is necessary also to form some estimate of 'confidence limits' on the sample means, to know whether or not a difference is insignificant, small, or large, having regard to variations in system performance. 95% confidence intervals were estimated for the means, and are shown graphically on Figure 2 as 'spreads' in the means.

An approximation was used, since the original responses are from a super-position of several rather different distributions (especially on the DEC) and are possibly not strictly independent (because of synchronization effects, etc.). The sample mean response time is assumed to be normally distributed (though the original responses clearly are not) by the 'Central Limit Theorem' [11]; the samples involved several hundred responses a run. The T-distribution [12] was therefore used. For example, the DEC responses in Figure 2(a) numbered 367 readings, with mean 5.2 and standard deviation 12.4. The estimated standard deviation of the mean is superficially $12.4/\sqrt{367} = 0.64$; the 95% confidence interval for the mean is estimated as $5.2 \pm \delta$ where $\delta = (12.4/\sqrt{367})^* \, T_{.025,366}$ and $T_{.025,366} = 1.97$ so $\delta = 1.27$.

This method is less elaborate than that described by Mamrak and DeRuyter [13], it but shows when the differences in sample means from different computers are statis-



(a) Script with Text Editor



(b) Script in Interactive BASIC

Figure 2. Sample Response-Time Distributions at 60 Users

tically significant. Having confirmed this, ratios of mean response times were used for scoring this aspect of system performance. A 2-stage procedure as described in [13] was impractical since our analysis could not start until after the tests were complete.

The measured cycle times were always effectively normally distributed, with a standard deviation smaller than the mean, for example 654s mean, standard deviation 13s, with 23 readings, from one typical run. The same procedure was followed for estimating confidence intervals. The smaller number of samples is acceptable since the distribution is essentially normal.

7. The Usefulness of Benchmarks

It is quite common for such benchmarks of computer systems to cost thousands of dollars, at least in staff costs and computer resources. The results of such a performance evaluation cannot show which system is best when the eventual workload is as unpredictable as that in a 'public' time-sharing system. Such a performance evaluation can only provide answers to relatively specific technical questions, which presumably are considered relevant to the overall assessment.

What we did succeed in doing was establishing approximately how powerful a machine and what memory size, etc. would be needed from each vendor, in order to obtain comparable and acceptable performances from the various proposed systems. This did force a revision of every proposal, with changes in price. We only obtained a rather limited impression, from the benchmarks, of how ultimately expandable the systems would be. I do not think they were very successful at that. The margin tests were probably not worth running.

We had not given enough thought beforehand to the mechanics of analysing the benchmark scripts. This was due partly to the difficulty of finding out in advance what the various stimulators and other facilities would provide. It would be helpful if vendors could supply descriptions and sample outputs to customer benchmark teams at an early stage.

Despite the difficulties, however, there is no doubt that the benchmarks were valuable as a component of the entire selection process.

Opinions expressed in this paper are those of the author, and do not necessarily represent the views of the Selection Committee, the Computing Centre, or the University.

I am indebted to the other members of the Selection Committee for many discussions concerning benchmarks and computer performance evaluations, and to vendor representatives for details of the various software and hardware systems concerned. Mr. John McHardy, as Chief Systems Programmer, played a major role in creating and supervising the benchmark programs.

References

[1] A.L. Scherr, An analysis of Time-shared Computer Systems, MIT Press, (1967).

[2] J.C. Adams & G.E. Millard, Performance Measurement on the Edinburgh Multi-Access System. EMAS Report #7, University of Edinburgh (1975).

[3] H. Rodriguez, Measuring User Charactermistics on the Multics System. Laboratory for Computer Science, MIT, Report MIT/CSC/TM-89 (1977).

[4] R.G. Bayly, The Design of Multi-Access Benchmarks. In Benchmarking Computer Evaluation and Measurement (ed. N. Brenwell), J. Wiley & Sons, Toronto (1975), p. 66.

[5] Interactive Facility RFP for Chilton, Science Research Council (UK) (1976).

[6] D.J.M. Davies, POP-10 User's Manual. Report #25, Department of Computer Science, University of Western Ontario (1976).

[7] R. Prendergast, Manufacturers' Attitudes towards Benchmarking. In Benchmarking - Computer Evaluation and Measurement (ibid) p.47.

[8] L.A. Belady & M.M. Lehman, Characteristics of Large Systems. In Research Directions in Software Technology, MIT Press (to be published in 1978).

[9] M.V. Zelkovitz, Perspectives on Software Engineering. ACM Computing Surveys 10 (June 1978), 197-216.

[10] F.J. Corbato & C.T. Clingen, A Managerial view of the Multics System Development. In Research Directions in Software Technology (ibid).

[11] W. Feller, An Introduction to Probability Theory and its Applications, Vol. I, 3rd Edn., John Wiley & Sons, NY (1968).

[12] J. E. Freund, Mathematical Statistics, Prentice Hall, Inc., NJ, (1962).

[13] S. A. Mamrak & P. A. DeRuyter, Statistical Methods for Comparing Computer Services. Computer (November 1977) 10, 32-39.

PROBLEMS IN REMOTE TERMINAL EMULATION

Vijay Trehan

Digital Equipment Corporation

RTEs are being increasingly used in performance studies. Performance predictions based on RJE experiments usually have greater credibility than similar predictions based on simulation/analytical models. However, significant problems exist in the areas of workload characterization, workload emulation and subsequent data analysis. This paper discussed the problems in remote terminal emulation. Alternative approaches for resolving these problems are suggested (including pros and cons for each approach).

Key words: Benchmark; clustering; job initiation; job selection; multivariate stochastic process; multivariate regression analysis; performance measures; remote terminal emulator; scenario; steady state; system under test; workload characterization.

1.0 INTRODUCTION

Remote terminal emulation is an approach to the testing and evaluation of computer systems in which a driver is implemented external to and independent of the system being tested. The driver, called the Remote Terminal Emulator (RTE), applies a specified workload to the system such that it appears to the System Under Test (SUT) that it is connected to live terminals. The objective of remote terminal emulation is to drive an SUT in a manner that is controllable, repeatable, economical and realistic. Figure 1 is a schematic of the flow of information in a remote terminal emulation experiment.

RTEs are being increasingly used in performance studies. Results from RTE experiments influence decisions all the way from the design to the sale of computer systems. Performance predictions based on RTE experiments usually have greater credibility than similar results based on simulation/ analytical models. However, significant problems exist in the areas of workload characterization, workload emulation and subsequent data analysis.

This paper discusses the problems in remote terminal emulation and suggests alternative approaches for resolving these problems (including pros and cons for each approach). Section 2 discusses the problems involved in defining benchmark workloads which are "representative" of "real" workloads. Section 3 details the problems associated with "correctly" generating a benchmark workload using a RTE. Section 4 describes the problems in analyzing performance data from RTE experiments and deciding which of two SUTs is "better". Section 5 outlines some future trends in the computer industry and the resulting impact on RTEs.

2.0 WORKLOAD CHARACTERIZATION

RTEs may be used for generating a wide variety of workloads. Generally, the workloads fall into the following generic classes.

1. Remote batch (RB)

2. Timesharing (TS)

3. Transaction processing (TP)

4. Real time processing (RT)

FLOW OF INFORMATION IN

RTE EXPERIMENTS

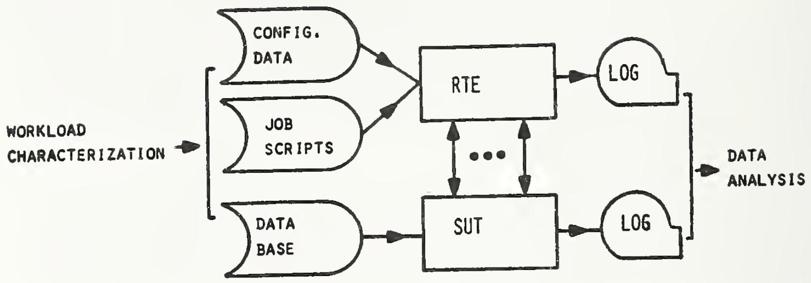


Figure 1

WHY FUNCTIONAL FEATURE SET?

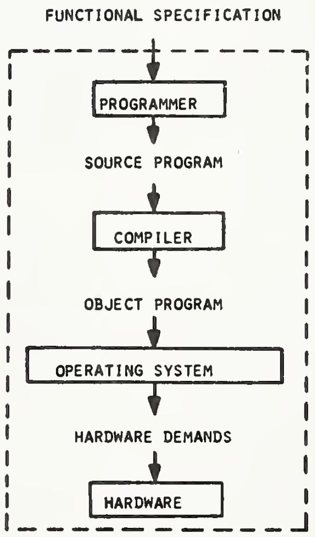


Figure 2

There are some RTE issues which are specific to each of the above environments. However, most of the problems discussed in this paper apply across all workload types.

A majority of RTE workloads used today are selected in an ad hoc manner. The results of RTE experiments are very sensitive to the workloads used. It is possible to make one SUT look better/worse than another SUT by simply changing the workloads used. Thus it is not sufficient to apply the same workload to the systems being compared. The only meaningful way to compare the performance of systems is to use workloads "representative" of the real workload environment.

One way for a RTE to drive a SUT is by using a trace of the real workload (as is done in trace driven simulation models). An advantage of using a trace is that it guarantees a high degree of representativeness. Among the disadvantages of using traces are that they are system dependent and rather voluminous. A more convenient approach is to use an abbreviated workload or benchmark representative of the real workload. The workload definition / generation capabilities desired in a RTE are directly related to the manner in which the benchmark workload is defined. The rest of this section will focus on the problems in obtaining representative benchmark workloads.

2.1 Measurement/Analysis Of Real Workloads

In order to construct representative benchmark workloads it is first essential to measure and analyze the characteristics of real workloads.

2.1.1 Selection Of A Unit Of Work

The unit of work used for characterizing a workload may be either a class of jobs, a job or a jobstep. A job typically refers to a sequence of one or more jobsteps which together perform some function for a computer user. A jobstep refers to a complete specification to the computer system of a task to be performed with no further interference from the user. If the exact sequence of jobsteps within a job are known a priori then it is adequate to use individual jobs as the unit of work for characterization purposes (this may be true in transaction processing environments). However, consider the case when the characteristics of individual jobsteps, within a job, are not known a priori. By characterizing the workload at the job level, information on the characteristics of individual jobsteps is not available when constructing the benchmark workload and thus may be misrepresented.

2.1.2 Characterizing A Unit Of Work

A general model for an individual unit of work may be represented as follows. See reference [1].

$$J = (X,T,L,F) \qquad (1)$$

where, J = unit of work (e.g. job, jobstep)
X = vector of features used used to characterize J (resource utilizations/ functions performed)
T = arrival time of J
L = location where J originated (e.g. terminal number)
F = flag denoting the generic environment of J (e.g. TS, TP)

The characteristics of each job/jobstep include the programs used and the databases accessed. Very little work has been done in the area of characterizing databases. Most examples in this paper will deal with the characterization of programs used by a job/jobstep.

The feature set (X) used to characterize J may be composed of system dependent features (e.g. CPU time, I/O time) or system independent features (e.g. compiles, edits). A functional feature set is desirable because it enables a system independent characterization of the workload. A feature set composed of hardware utilizations, on the other hand, is affected by the characteristics of the programmer, compiler, operating system and the hardware for a particular system (see Figure 2). In addition, resource utilization data includes the effect of delays caused by the terminal operator, the terminal and the communication interface. In general, a feature set as close as possible to a functional feature set is desirable. The number and type of features which form a necessary and sufficient set for characterizing a workload varies with the type of workload and the objective of the study. Workload dynamics (across systems) and user adaptation to systems are areas in which little work has been done. It is unclear to what extent it is even possible to characterize workloads in a system independent manner.

If the arrival time (T) of J is not observed then it is not possible to accurately represent job/jobstep sequences or inter-arrival times in benchmark workloads. Also, if T is not observed it is not possible to study the variation of load over time (e.g. peak periods). In timesharing environments it has shown that the presence of certain commonly occuring jobstep sequences (e.g. edit-compile-execute) have a significant performance impact. See reference [4]. Figure 3 is a simple example illustrating how correlated arrival times can significantly alter performance.

If the location (L) where J originated is not used then the assumption is made that the workload is homogeneous across all terminals (i.e. all terminals handle workloads with the same characteristics). In a typical real system, however, there are groups of terminals dedicated to certain tasks. For instance, if 50% of the terminals always execute compute bound FORTRAN jobs and the other 50% of the terminals always execute I/O bound COBOL jobs then it is essential to represent this fact in the benchmark workload (see Figure 4A). Having all the terminals executing a mix of 50% FORTRAN and 50% COBOL jobs will produce erroneous results (see Figure 4B).

2.1.3 Measuring Real Workload Data

The next step is to measure the characteristics of a real workload. The measurement area poses several problems. The proposed processing environment may not exist - for instance when a SUT is being designed for a new market. In such cases it is necessary to extrapolate the characteristics of the proposed workload from some existing similar environment. The representativeness of such projections is hard to verify. Even when the workload environment exists the data usually available (e.g. accounting/log) falls far short of what is desirable. Standard accounting/log data provides system dependent resource utilizations with no information on arrival time or terminal of origin of individual jobs/jobsteps. By using such data it is not possible to reconstruct the load as it was submitted at the remote terminals.

2.1.4 Analysis Of Real Workload Data

The next step is to analyze the measured data. In its general form the analysis problem involves the study of the multivariate stochastic process (X,T,L,F) - refer to section 2.1.1. Most current approaches decompose this problem into several simpler problems. Typically, the resource utilization / functional characteristics are studied separately from the arrival time characteristics of the jobs/jobsteps. Such a decomposition assumes a stationary stochastic process. Multivariate statistical techniques such as clustering over the feature space (X) have been used to study the time independent characteristics of the workload. Data points in the feature space tend to form clusters and it has been observed that functionally similar workloads cluster in a similar fashion.

WHY JOB ARRIVAL TIME?

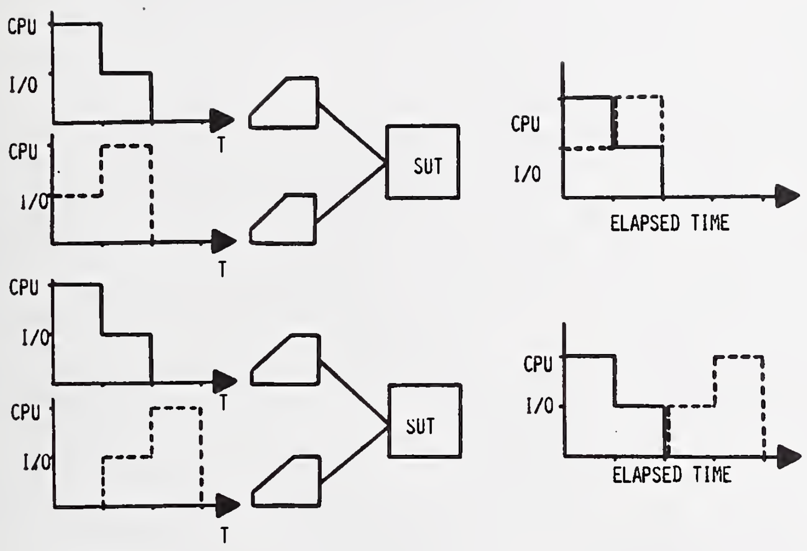


FIGURE 3

WHY WORKLOAD PER TERMINAL?

J_1 = COMPUTE BOUND JOB

J_2 = I/O BOUND JOB

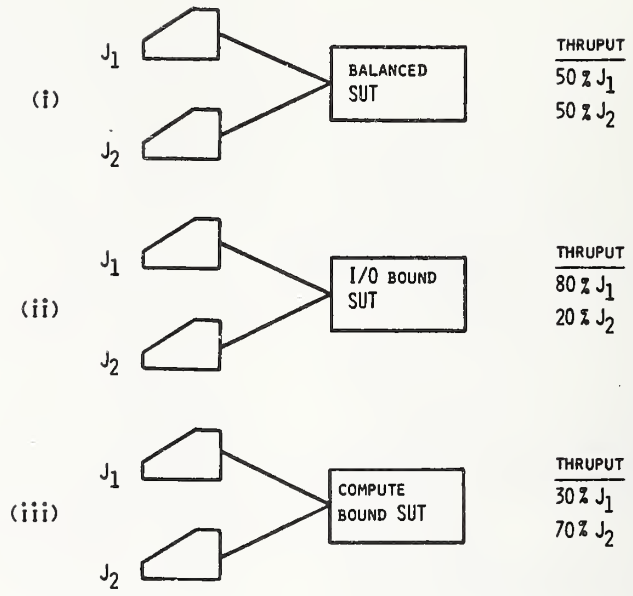


FIGURE 4A

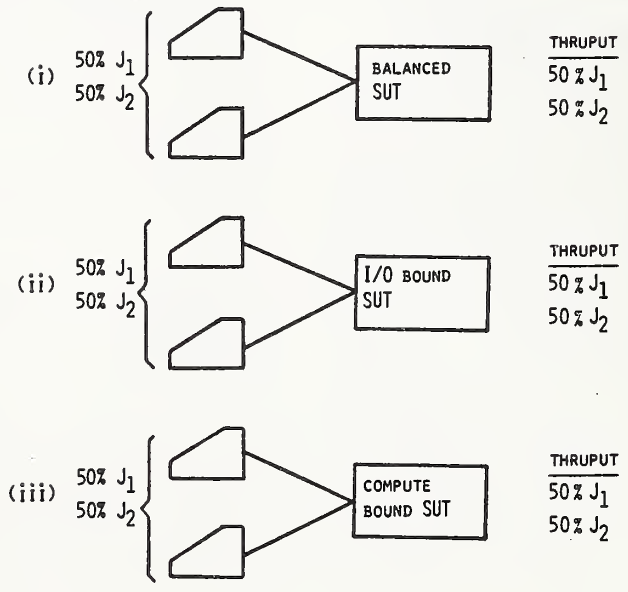


FIGURE 4B

For details of clustering techniques (e.g. scaling, dimensionality, distance measures, transformations, algorithms, robustness) see reference [5]. Time dependent characteristics may then be analyzed for each cluster. Job/jobstep sequences may be modelled using Markov transition probabilities. Transition probabilities may be specified for a level of dependency D (i.e. the next job/jobstep is chosen based on the preceeding D jobs/jobsteps) - see section 3.1.2. Recent work at the University of Maryland and Bell Laboratories has shown that for timesharing systems D>1 is necessary to accurately represent the workload. Job inter-arrival times (within or across clusters) may be fitted to a negative exponential or other statistical distribution.

2.2 Developing Benchmark Workloads

Assume that the real workload has been characterized as a set of weighted clusters in the feature space. Jobs in each cluster have some functional characteristics in common. Inter-arrival time distributions and Markov transition probabilities are specified for each cluster. Several approaches for constructing benchmarks have been used in the past such as having one benchmark job representing the centroid of each cluster. Benchmark workloads may be composed of synthetic jobs or real jobs (obtained by sampling the real workload).

Establishing the representativeness of benchmark workloads poses several significant problems, especially if the real workload is non-repeatable or non-existent (see Figure 5). It is desirable that the representativeness of benchmarks be verified on more than one system to ensure that the benchmark is not system dependent. Certain characteristics of the benchmark workload and the real workload will match by definition. What additional characteristics (e.g. job elapsed time, resource utilizations) must match for the benchmark to be considered representative? The example in Figure 6 shows why elapsed time of job(s) is not enough for verification. The example in Figure 7 shows why elapsed time of job(s) and resource utilization(s) together are not enough for verification. Depending on the resource usage profile, the result of introducing a new job may be quite different. Clearly, comparing the resource usage profiles of the real workload and the benchmark workload is not an easy task. Thus verifying the representativeness of benchmark workloads remains a problem.

2.3 Developing Benchmark Scenarios

Finally, after the benchmark workload has been constructed and validated it must be converted into a system independent description (called a scenario). For this it is essential to develop a system independent scenario language. Scenario elements will represent individual functions. The scenario elements required to describe timesharing workloads may be different from those required to describe transaction processing workloads. Some work in this area is being done by the government (GSA). The proposed scenario language is plain English.

Prior to each RTE experiment the benchmark scenario must be translated into scripts. Although the use of scenarios introduces additional overhead, it promotes system independent description of workloads.

Assuming that workloads are, to a large extent, a characteristic of the environment in which a computer system operates rather than of the system itself, it should be possible to set up a library of representative benchmark scenarios for performance studies.

3.0 WORKLOAD EMULATION

An RTE must represent the significant characteristics of the benchmark workload, the terminal operator, the terminal and the communication interface (see Figure 8).

REPRESENTATIVENESS OF BENCHMARK WORKLOADS

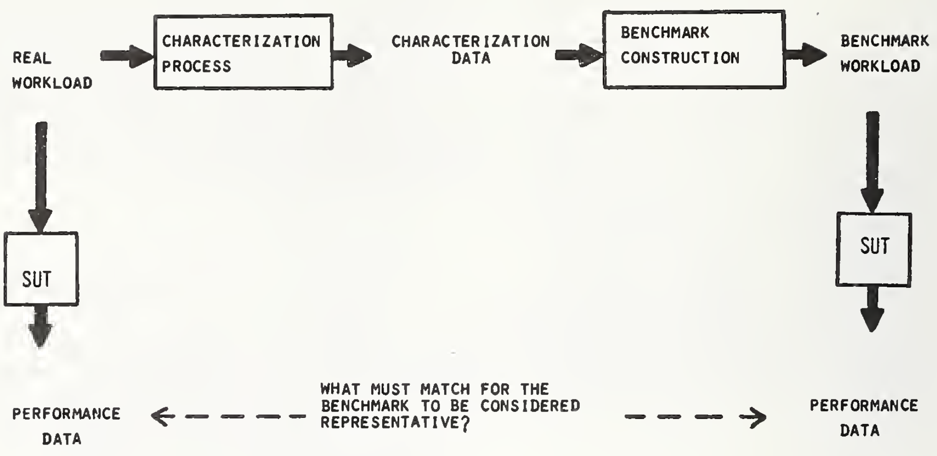


Figure 5

WHY RESOURCE UTILIZATIONS SHOULD MATCH?

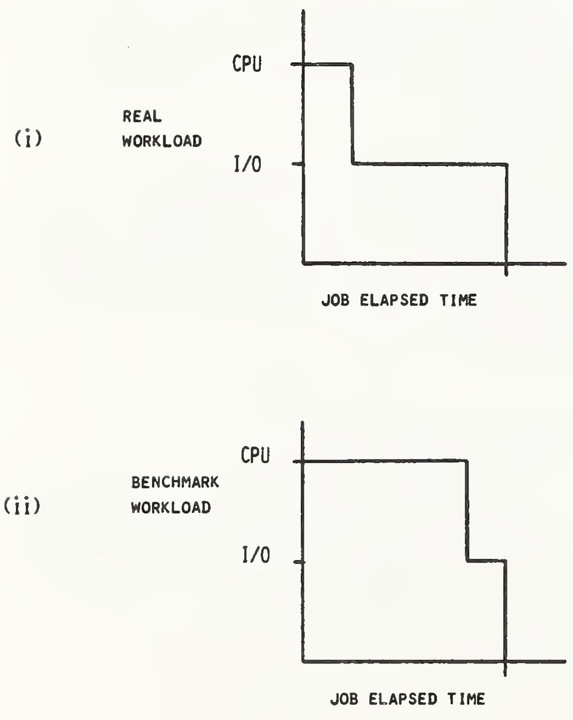


Figure 6

WHY RESOURCE USAGE PROFILE SHOULD MATCH?

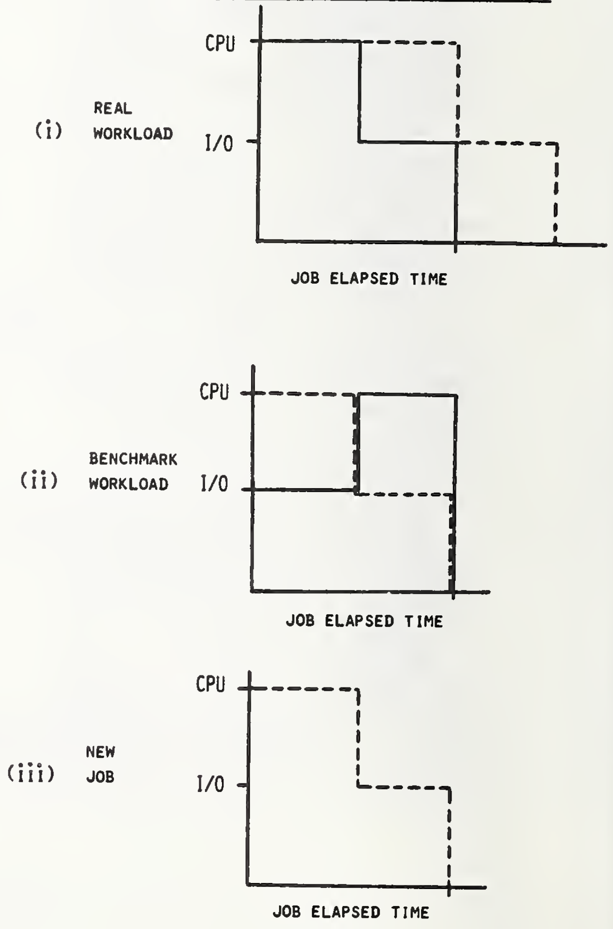


Figure 7

WHAT MUST BE EMULATED IN A RTE?

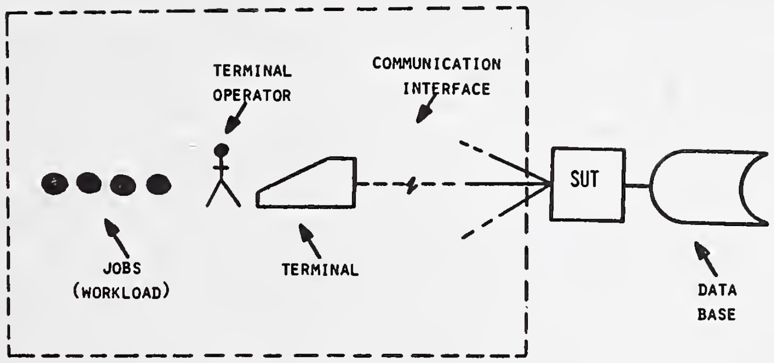
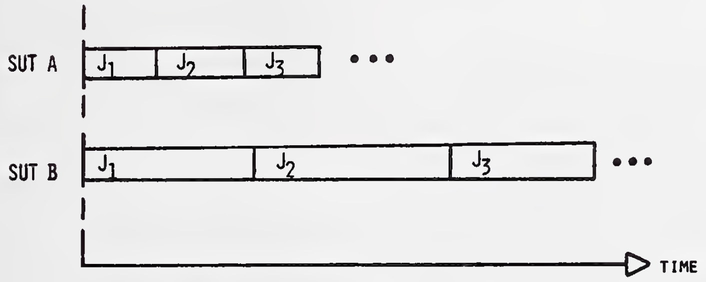
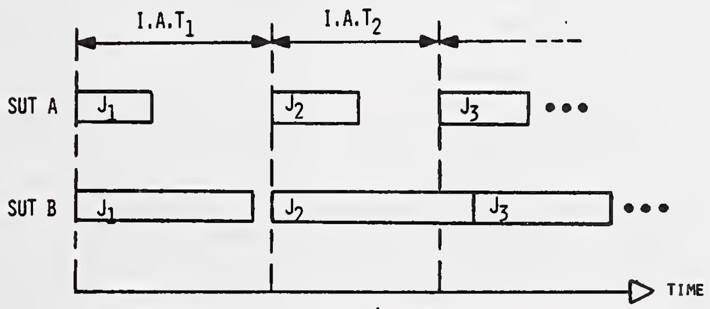


FIGURE 8

JOB INITIATION



REPLY BASED JOB INITIATION (LOADING IS SYSTEM DEPENDENT)



INDEPENDENT JOB INITIATION (LOADING IS SYSTEM INDEPENDENT)

FIGURE 9

3.1 Workload Representation

Benchmark job descriptions are input to the RTE in the form of job scripts. Each job script is an SUT-RTE dependent translation of a benchmark job scenario. Scripts include message texts, job scheduling and other control information.

3.1.1 Message Text Information

Scripts, as defined by RTE users, do not usually include communication control characters (these are introduced by the RTE software). The following are three alternative approaches for representing message texts in job scripts (note: all messages originating at the RTE will be referred to as "queries" and all messages originating at the SUT will be referred to as "responses").

1. Only query texts.

2. One expected response text per query.

3. Several possible response texts per query.

In most prompt oriented systems the operator waits for a prompt from the system before proceeding with the next query. Typically, the prompt is a special character at the end of the last response message from the SUT. It is essential in an RTE to know the point in time when a real operator would start the keying or thinking for the next query. If the first approach is used (i.e. RTE does not scan response texts) and if the SUT does not have a uniform way of informing the terminal that it is ready for the next query (e.g. by sending a keyboard enable character) then the SUT operating system must be modified to send a specific prompt character, recognized by the RTE, when it is ready for the next query. While the use of special prompt characters simplifies the "end of response" search, it is possible for the RTE and SUT to get out of synchronization. This occurs when the SUT sends an unexpected response and the RTE interprets it as a correct response by virtue of having received a valid prompt character.

The second approach actually scans the response text and matches it with the expected response text included in the script. Recovery procedures are initiated if the expected response is not received within a certain time period or a certain number of retries. The advantage of this approach is that the SUT operating system does not have to be modified and there is lesser likelihood of the SUT and RTE getting out of synchronization. A special case of this approach is when a different SUT prompt may be specified for each query. The chief drawback of this approach is that it does not provide for systems in which several valid responses to a query are possible or systems in which unsolicited messages are possible. The script preparation for this approach is more complicated than in the first approach.

In the third approach, several response texts are possible for each query. RTE actions are contingent on the nature of the SUT response to a query. This approach truly emulates an interactive environment. This approach is the hardest to implement due to the large number of query-response combinations. Performing scan and match operations on all response texts greatly increases RTE processing time per message and typically involves the use of large tables and considerable intelligence in the RTE. If all SUT responses have a header which contains response codes, then these response codes can be used by the RTE to make contingency decisions. The ability to send response codes must be designed into the SUT. Another variation of this approach is a SUT which sends any one of a set of valid prompts recognized by the RTE. The RTE examines the prompt received and makes the appropriate contingency decisions.

Another important feature relating to message texts is the capability to vary query texts for consecutive executions of the same script. For example, if a transaction accesses a database using as a key the customer ID, it would be desirable to change this ID for successive executions of the transaction. Similarly, a job doing a edit, compile and execute sequence may use different COBOL source files on successive passes. Text variation may be done using values obtained from tables/databases or using random numbers, line numbers, terminal numbers, response texts etc. Note that the nature of text variation must be outlined by the benchmark workload.

3.1.2 Job Scheduling Information

Given a set of job scripts to run on a RTE, there are two basic problems in the scheduling of jobs. The first deals with the initiation of jobs and the second deals with the selection of jobs. Scheduling may be done on a per terminal/per line/per SUT basis (depending on how the workload is characterized). Similar scheduling can be applied to the jobsteps within each job.

3.1.2.1 Job Initiation

The common method of scheduling arrivals of jobs employs a randomized inter-arrival time. Two alternative mechanisms exist for deciding when to initiate the next job script (see Figure 9).

1. Reply based job initiation.

2. Independent job initiation.

In the reply based method the RTE initiates the timing of inter-arrival delays only upon completion of the preceeding job. The chief drawback of this method is that the job initiation depends on the SUT response time characteristics and therefore cannot be held constant across systems. However, this method is useful for saturation loading (i.e. zero inter-arrival time).

In the independent method the RTE generates job arrivals (using randomized inter-arrival times) without regard to whether the previous job is still in progress. If the previous job is still in progress when an arrival occurs, the new job must wait on a queue. At the completion of the experiment queue statistics are reported along with response time and other statistics. This method explicitly provides a means for holding the load constant across SUTs.

Script initiation on successive terminals is usually staggered (in time), at the start of an experiment, to avoid any lockstep/cyclic loading.

3.1.2.2 Job Selection

In addition to deciding the initiation time for the next job, a specific job must be selected for execution. Job selection mechanisms may be broadly classified as follows.

1. Dynamic job selection.

2. Static job selection.

In the dynamic case the next job is selected from a set of jobs (on the fly) by one of the following methods.

1. Random - A uniform[0,1] random number is used along with a Markov transition probability matrix to select the next job to be executed (see Figure 10). The "D level dependency" Markov transition probability matrix is a general model for representing random job sequences. If there are N job types and S possible sequences of length D, then a SxN transition probability matrix defines the dynamic job mix. A commonly used special case is the simple job mix i.e. all jobs are independent (D=0, S=1). In timesharing environments the use of transition probabilities is more meaningful at the jobstep level. This is because there are commonly occuring jobstep sequences (e.g. edit-compile-execute) but successive users of a terminal are independent. Thus a simple mix may be specified for the jobs in a benchmark workload and a transition probability matrix

may be specified for the jobsteps within each job.

2. Initiation control - The next job is selected solely to maintain a job mix (or in the general case a transition probability matrix). See reference [3]. No random number is used to select the next job.

	Jobs 1	2	. . .	N
Target initiation mix	0.2	0.1	. . .	0.15
Observed initiation mix	0.18	0.12	. . .	0.1
Ratio observed/target	0.9	1.2	. . .	0.67

(Job with minimum ratio is selected)

The main advantage of this method is that it has been shown to converge to a steady state mix faster than the random method. For additional discussion on steady state issues see section 4.2. A disadvantage is that while this method maintains the job mix it may induce unrepresentative sequences of jobs (e.g. cyclic initiation patterns). This disadvantage can be minimized if initiation control is used along with a "D level dependency" Markov transition probability matrix. Then sequences up to depth D will be represented correctly.

3. Profile control - In this method the next job is selected solely to maintain an "active job mix". See reference [3]. No random numbers are used to select the next job. If 20% of the terminals are always doing compute bound FORTRAN jobs and the remaining 80% of the terminals are always doing I/O bound COBOL jobs then the active job profile of FORTRAN : COBOL will reamin constant at 1:4. Note, here, that the definition of "active" includes all intervening terminal operator delays, terminal delays and communication delays during the execution of a job.

	Jobs 1	2	. . .	N
Target profile	0.1	0.2	. . .	0.1
Current profile	0.1	0.18	. . .	0.2
Ratio current/target	1.0	0.9	. . .	2.0

(Job with minimum ratio is selected)

The main advantage of this method is that it converges to steady state faster than the random method. It is possible to run experiments with varying numbers of terminals without having to reassign job scripts to individual terminals (as long as the profile is known). The profile is maintained even if a line goes down. Profile control is not suitable where a job mix is associated with each terminal (or groups of terminals).

In the case of static selection the next job is obtained from a file of job scripts (the entire sequence of jobs is determined a priori). Static selection methods fall under the following headings.

1. Random - The same methods used in the dynamic random case may be applied with similar results. One of the disadvantages of the static random approach is the large storage requirements for the script file. The advantage of this approach over the dynamic random approach is that several features which are hard to implement in the dynamic approach (e.g. query text variation, alternate response texts) can be included in the scripts by using a preprocessor to generate the script

RANDOM JOB SELECTION

- MARKOV TRANSITION PROBABILITIES (DEPTH=D)

JOBS/JOBSTEPS = $J_1, J_2 \ldots\ldots\ldots J_N$

D = 0 SIMPLE MIX

J_1	J_2	. . .	J_N
0·1	0·2	. . .	0·1

FIGURE 10A

D = 1 TRANSITION PROBABILITIES

NEXT LAST	J_1	J_2	. . .	J_N
J_1	0·2	0·5	. . .	0·0
J_2	0·1	$\ddots$		
.	.			
.	.			
.	.			
J_N	0·15			

FIGURE 10B

D = 2 TRANSITION PROBABILITIES

NEXT LAST TWO	J_1	J_2	. . .	J_N
J_1J	0·1	0·1	. . .	0·2
J_1J_2	0·15	$\ddots$		
.	.			
.	.			
.	.			
J_NJ_N	0·0			

FIGURE 10C

file.

2. Loop - A sequence of job scripts is constructed by the RTE user (typically without the aid of any preprocessor). This sequence of job scripts is repeated in a loop during the RTE experiment. Control over job mix and sequence is limited by the number of jobs in the loop.

3.2 Terminal Operator Representation

The principal features of the terminal operator represented in the RTE are the think time and the typing rate. In most interactive environments a majority of the duration of a terminal session is spent in thinking and typing. Thus the load generated by a terminal is limited by the speed of the terminal operator. This only emphasizes the need to have accurate human factors models for these times.

There are several approaches to the modeling of think time and typing time. Take the example of think time. It may be represented by a random variable. Different statistical distributions may be used. Think time may also be modelled as (constant time+ random component). Think time may be dependent on the job being performed. Think time may be separated into overlapped and non-overlapped think time. Overlapped think time is that portion of the think time which is allowed to overlap the SUT response time. The non-overlapped think time cannot start until a complete response has been received from the SUT.

3.3 Terminal Representation

The principal characteristics of an emulated terminal represented in the RTE are the communication interface (see next section) and any significant delays that occur in the terminal. Correct representation of input delays (e.g. card reading), output delays (e.g. display, printing) and terminal processing delays is critical when these delays may limit the load generated by the terminal or influence the response time (as it is perceived by the operator). Certain special features such as "type ahead" are usually harder to implement in a RTE.

3.4 Communication Interface Representation

A variety of communication equipment (e.g. lines, modems, bridges, concentrators, multiplexors, communication controllers, network nodes) usually connects terminals to a host computer system. RTE experiments must reproduce the significant communication delays (e.g. line contention, transmission time including store and forward, time to establish carrier on modulated lines) which occur in the real system. There are three alternative approaches for representing communication equipment in RTE experiments.

1. Ignore communication equipment.

2. Physically include communication equipment between the RTE and SUT.

3. Emulate communication equipment in the RTE software.

Omission of communication equipment in systems that are communication bound will result in erroneous performance data. Physically including all communcation equipment between the RTE and the SUT requires a lot of hardware for large configurations. Emulation of communication equipment requires complicated RTE software and increased RTE processing time per message. Often a combination of the above approaches is most desirable.

Correct representation of multidrop / multiplex mode lines is necessary to reproduce line contention and SUT overhead. Some typical RTE-SUT interfaces for a multidrop line are shown in Figure 11. The communication protocols, link types, link speeds etc. at the SUT interface should be the same

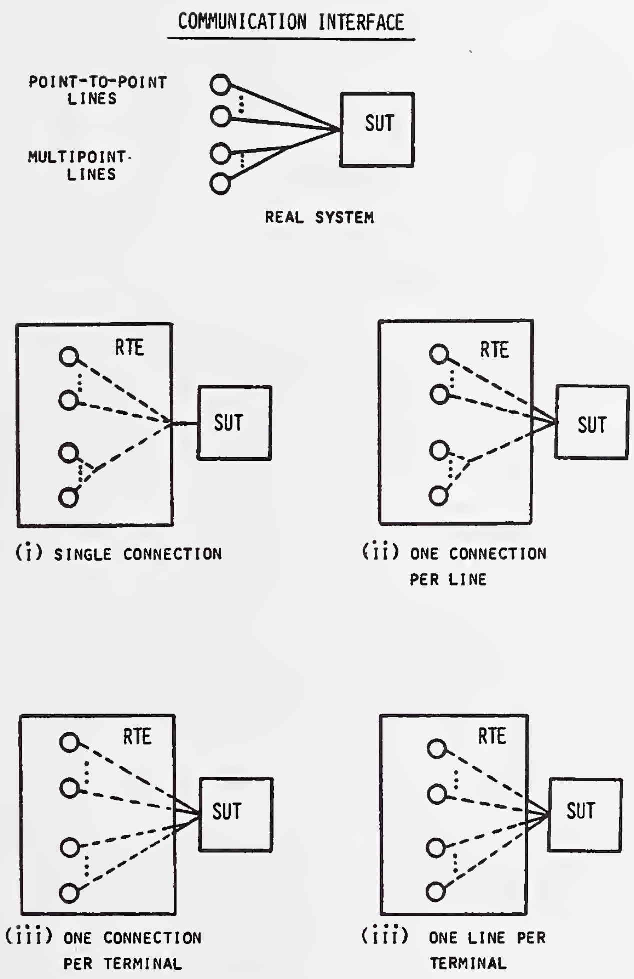

COMMUNICATION INTERFACE
POINT-TO-POINT LINES
MULTIPOINT LINES
SUT
REAL SYSTEM
RTE
SUT
(i) SINGLE CONNECTION
RTE
SUT
(ii) ONE CONNECTION PER LINE
RTE
SUT
(iii) ONE CONNECTION PER TERMINAL
RTE
SUT
(iii) ONE LINE PER TERMINAL

FIGURE 11

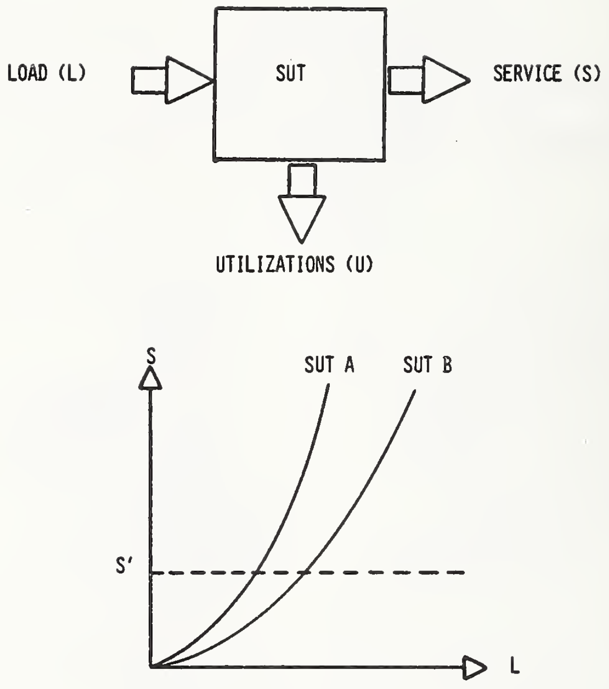


Figure 12

as in a real system configuration (in order to exercise the appropriate module of the SUT communication software). For this reason, having a single high speed trunk between RTE and SUT is not desirable. Having one connection between RTE and SUT for each line in the real system configuration is usually the most desirable. The task of emulating multidrop / multiplex mode lines is often simplified by using SUT line control software, with some modifications, in the RTE (for this the RTE must be based on a processor compatible with the SUT processor). Having one line between RTE and SUT per terminal is easier to emulate but requires too many communication lines for large configurations and is not representative of real systems which use multidrop / multiplex mode lines. The cost of RTE experiments is directly related to the number of lines.

4.0 DATA ANALYSIS

How performance data from RTE experiments is collected and analyzed directly impacts the resulting decisions on which of two systems is better. Service (S) is measured in terms of the typical performance measures such as elapsed time, response time etc. Load (L) is measured in terms of job thruput. There is a minimum (or maximum) level S' for the service to be considered acceptable. Figure 12 schematically shows the comparison of performance of SUT A and SUT B.

4.1 Performance Measures

The performance measures suitable for a particular study depend, largely, on the nature of the user environment in which the SUT will operate. Usually, very little effort is put into the choice and definition of appropriate performance measures. environment in which the system will operate.

To illustrate the need for clearly defining performance measures consider the specific case of response time. Performing a given function on different SUTs may result in a different number of queries/responses (see Figure 13). Sometimes there are several different ways to perform a function on the same SUT (e.g. VMS commands allow different levels of prompting). A single query may result in several responses (some of which may simply be SUT acknowledgements for receiving the user's query and not a true response). Given all this, it is unclear as to what constitutes response time for a given function. A system independent definition of reponse time is necessary (vs. a last-bit-in to first-bit-out type definition). The only definition of response time which has any performance significance is one that provides some measure of terminal operator inconvenience (caused by a delayed response). This, of course, is a complex function of the characteristics of the system and the terminal operator.

Assume the simple case where each function results in one query and one response. The response, perceived by the terminal operator, is made up of terminal delays, communication delays and SUT delays (see Figure 14). In some systems the communication delays will dominate the response time. In such a situation the difference in response time between two SUTs may be due to the difference in characteristics of the communication interface and not the SUTs. In order to make a fair comparison it is necessary to use representative communication interfaces in RTE experiments. It should be pointed out that considerable optimization is possible in a complex communication interface. Communication delays can be significantly increased / decreased by simply reconfiguring the communication interface (e.g. line speeds, terminals per line).

DEFINITION OF RESPONSE TIME?

FUNCTION - COPY THE CONTENTS OF FILE
X'IN INTO THE FILE X'OUT

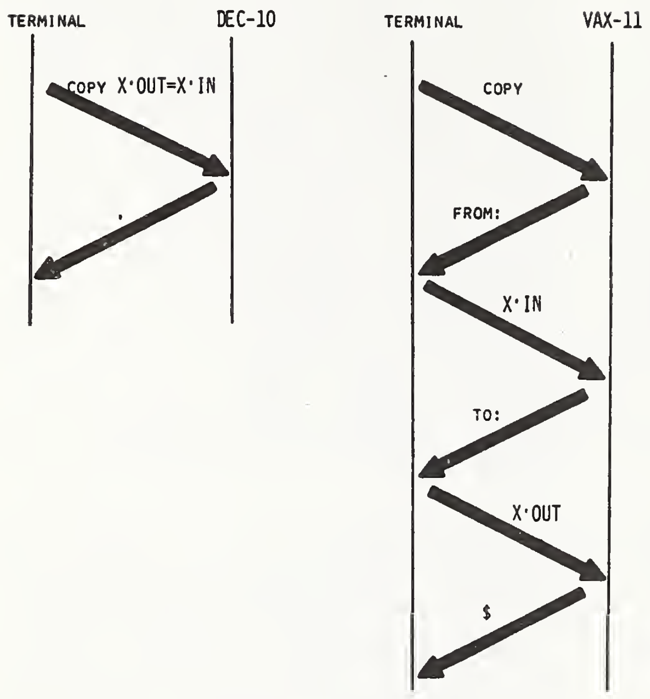


FIGURE 13

BREAKDOWN OF RESPONSE TIME

O = OPERATOR DELAYS
T = TERMINAL DELAYS
C = COMMUNICATION DELAYS
S = SUT DELAYS

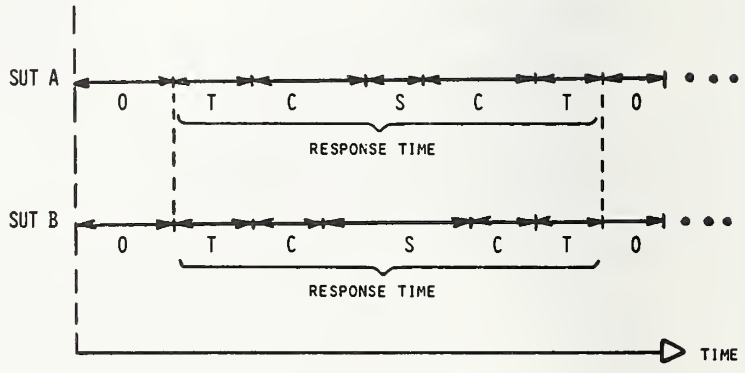


FIGURE 14

4.2 Steady State Issues

Typically, the performance data for the initial T' time units (transient state) of a RTE experiment is discarded. This represents the warm up phase when the values of the performances are stabilizing. (See Figure 15). Discarding this data results in a lower variance of the performance measures and a shorter overall duration of the experiment. Data collected from T' to T" time units (steady state) is used to obtain statistics on performance measures.

The values and statistical distributions of the performance measures are very sensitive to T' and T" (unless the experiments are run for very long periods of time). When comparing the performance of different SUTs simply comparing the mean values of performance measures is not adequate. The statistical distributions of performance measures must be taken into account in the analysis. This may be done by considering the confidence intervals associated with the performance measure(s). The use of confidence intervals makes the task of comparing SUTs more complicated. Figure 16 shows two SUTs with overlapping confidence intervals. It should also be pointed out that the confidence intervals are primarily dependent on the number of observations (sample size) and statistical distribution of the performance measures and not on the absolute duration of the steady state period.

4.3 Comparing The Performance Of Systems

Assume that all the performance measures for two or more SUTs have been measured, at different levels of loading so that a fair comparison can be made. In the general case there will be N different job types and M performance measures and P parameters describing the configuration (hardware/software) of the SUT. Performance(S) is represented by a multivariate response surface which may be modelled as a function(F) of the load(L) and the SUT configuration(C) as follows:

$$S = F(L,C) + E \qquad (2)$$

where, S = MxN dimensional vector representing service (e.g. elapsed time and response time for each job type)

L = N dimensional vector representing load (e.g. jobs per second for each job type)

C = P dimensional vector representing SUT configuration (e.g. processor type, memory size, operating system version, number of disks)

E = MxN dimensional vector representing the experimental error

Note that the above model assumes a stationary response surface. Multivariate regression analysis may be used to fit different functions(F) to the data observed from RTE experiments. As M,N and P get larger this problem gets very complicated. Deciding which of two systems performs better is a complex data analysis problem involving the comparison of two response surfaces F(L,C1) and F(L,C2) where C1 and C2 are the configuration vectors for the two systems. The problem is made more complex by the fact that there is a confidence volume associated with each point on the response surface.

Consider the simple case with a single performance measure and two job types. Assume that the job mix is constant at different levels of loading. The load-performance relationship may be represented by two sets of curves as shown in Figure 17. It is likely that one jobtype will do better on SUT A and worse on SUT B. It is not clear whether SUT A is "better" than SUT B or vice-versa. Some kind of objective function needs to be defined assigning weights to each job type - performance measure combination. See reference [14] for a cost-benefit approach for comparing

STEADY STATE

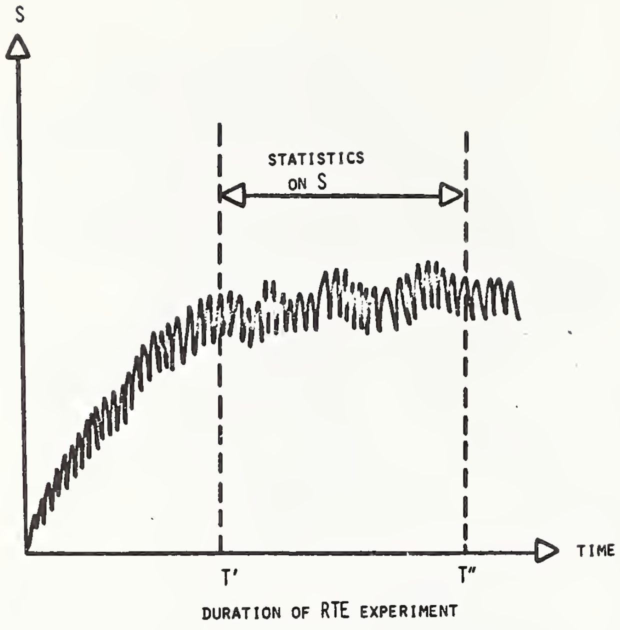


Figure 15

CONFIDENCE INTERVALS

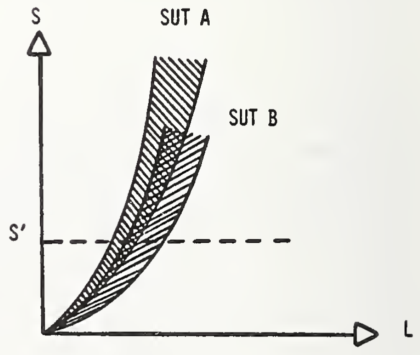
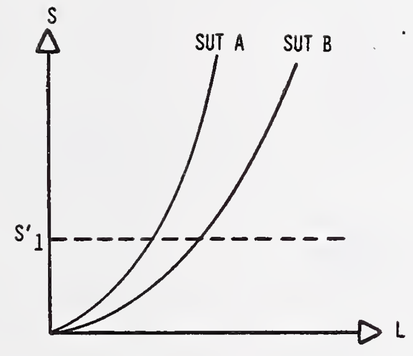
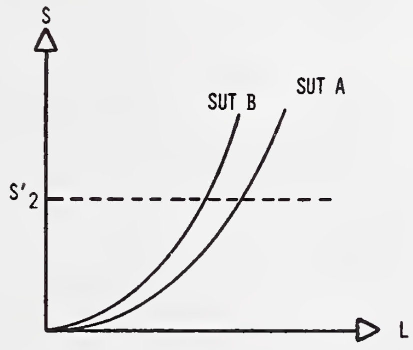


Figure 16



(i) job type 1



(ii) job type 2

Figure 17

the performance of systems. Most of the analysis work done today ignores the multivariate nature of the problem.

5.0 FUTURE TRENDS

So far this paper has been limited to the study of configurations with one RTE connected to one SUT. However, configurations requiring multiple RTEs may be fairly common. For instance, when a large number of terminals are to be emulated or when several RTEs are used to load the different nodes of a network (see Figure 18). There are several problems associated with synchronizing the load generation and data analysis for the different RTEs involved in such an experiment.

The use of distributed processing, intelligent terminals and computer networks is likely to have a significant impact on remote terminal emulation. It is not practical to configure large computer networks in a lab (just as it is not practical to configure large numbers of terminals). RTEs may be called on to emulate host-to-host lines and computer network nodes (see Figure 19).

The government is currently working on setting up RTE guidelines. Computer system vendors will be required to have RTEs meeting these guidelines to be eligible to bid on government contracts. See references [7],[8]. In the future, other large customers may have similar requirments for RTEs.

The trend appears to be towards the use of table driven RTEs which use a uniform approach for emulating timesharing, transaction processing, remote batch, realtime and distributed processing environments.

6.0 GLOSSARY

1. BENCHMARK -A set of computer programs and associated databases tailored to represent some real workload environment.

2. JOB -A sequence of one or more jobsteps which together perform some function for a computer user.

3. JOBSTEP -A complete specification, to the SUT, of a task to be carried out without further interference from the user.

4. QUERY -A message originating at the terminal. Communication protocol messages are not included.

5. REALTIME PROCESSING - Characterized by event driven data sensors providing stimuli to (canned) application programs which respond with stimuli for the purpose of controlling some realtime process. Absolute response time constraints are usually necessary.

6. RESPONSE -A message originating at the SUT. Communication protocol messages are not included.

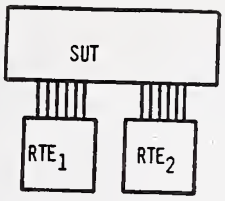
7. REMOTE-BATCH/BATCH PROCESSING - Characterized by the concurrent availability to the SUT of a complete set of input data for processing a given job. Execution of the job is not controlled in real time by the user.

8. REPRESENTATIVENESS -The ability of a benchmark workload to produce an identical sequence of resource demands on the SUT as the real workload.

9. RTE -A remote terminal emulator is a load driver implemented external to and independent of the SUT. It applies a specified workload to the SUT such that it appears to the SUT that it is connected to live terminals.

MULTIPLE RTEs

(i)



(ii)

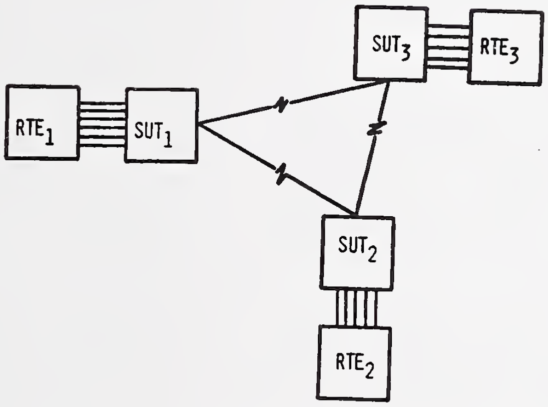


FIGURE 18

NETWORK SIMULATOR

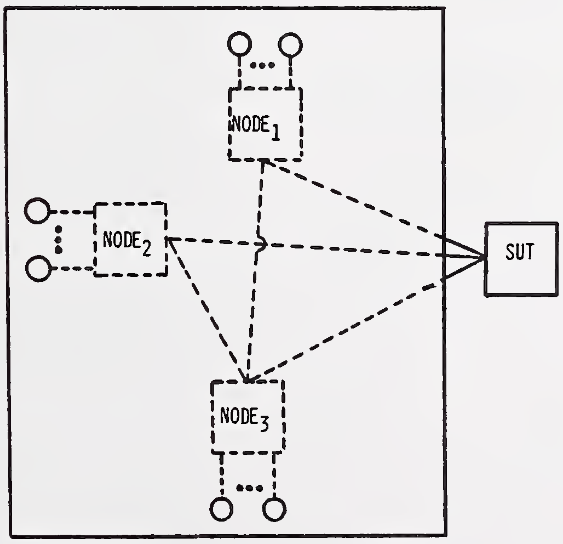


FIGURE 19

10. SCENARIO -A system independent description of a benchmark job.

11. SCRIPT -A SUT-RTE dependent translation of a benchmark job used to describe the job to the RTE. Scripts contain message texts and control information.

12. SUT -System under test.

13. TIMESHARING -Characterized by multiple interactive terminals (typically character oriented) being used for interactive program development/testing, word processing and interactive problem solving.

14. TRANSACTION PROCESSING - Characterized by multiple interactive terminals (typically message/forms oriented) being used to drive (canned) application programs performing data entry and database inquiry/update. Each application program services a specific commercial transaction.

15. WORKLOAD -The workload on a SUT may be defined as the entire time sequence of jobsteps entering the SUT during some interval.

7.0 REFERENCES

1. Agrawala,A.K., Mohr,J.M., Bryant,R.M., "An Approach to The Workload Characterization Problem" , Computer,June 1976.

2. Watkins,S.W., Abrams,M.D., "Survey of Remote Terminal Emulators" , NBS Special Publication 500-4, April 1977.

3. Hyman,B., "Stability and Workload Definition for Timesharing Systems" , Draft Minutes of FIPS TG13 Meeting , July 1975.

4. Manochio,L.J., "Representing Workload Dynamics" Draft submitted for publication at 13th meeting of CPEUG,1977.

5. Anderberg,M.R., "Cluster Analysis for Applications" , Academic Press, New York (1973).

6. "Teleprocessing Network Simulator (TPNS)", Program Reference Manual, IBM SH20-1823-2.

7. "Remote Terminal Emulation Specification For Federal ADP Systems Procurement", Draft GSA Guidelines, scheduled for October 1978 release.

8. "Remote Terminal Emulation in Federal ADP System Procurements", Draft GSA Guidelines, scheduled for Oct 1978 release.

9. Conti,D.,"Workload Characterization and Benchmarking: Problems and Practices", Presentation at the First Digital Performance Evaluation Symposium, May 1978.

10. Potter,T.,"Specification of a Load Driver used in Benchmarking Timesharing, Realtime and Remote Job Entry Systems".

11. Turner,R.,Levy,H.,"Performance Evaluation of IAS on the PDP-11/70", Proceedings of the International Symposium on Computer Performance Modeling, Measurement and Evaluation, Cambridge, Mass., March 1976.

12. Hess,P.R.,Strauss,J.C.,"A Simulation Study of Timesharing Cost-effectiveness", SIGMETRICS/CMG III Conference, Washington D.C., October 1977.

13. Wright,L.S.,Burnette,W.A.,"An Approach to Evaluating Timesharing Systems: MH-TSS, A Case Study", ACM SIGMETRICS Performance Evaluation Review, January 1976.

14. Jain,A.K.,Potter,T.W., "Statistical Modeling of Computer Performance (A Cost-Benefit Approach)", Proceedings of the 12th Meeting of CPEUG, 1976.

15. Potter,T.W.,Jain,A.K., "Statistical Modeling of Computer Performance (Simple Program Mix)", Proceedings of the 9th,10th and 11th Meetings of CPEUG, 1974/75.

PREDICTION PART I: METHODS 1978

PREDICTION PART I: METHODS 1978

Samuel H. Fuller

Digital Equipment Corporation
Maynard, Massachusetts

The papers in this session deal with performance models of computer systems. The first three papers investigate unconventional, innovative approaches to modeling performance. Concepts from control theory, Petri nets, and schemas for concurrent processes are all explored in these papers. As with any new approach, there are many limitations and unresolved issues in applying these techniques to practical performance problems. However, an important function of a technical conference such as this is to provide a forum for discussion of new approaches and methods that require further development.

The last two papers in this section are a review and an extension to queueing models. While these papers dealing with queueing models may not be as controversial as the earlier papers, they discuss different facets of what has been to date our most useful analytic modeling technique.

A FORMAL TECHNIQUE FOR ANALYZING THE PERFORMANCE OF COMPLEX SYSTEMS

John Sanguinetti

Digital Equipment Corporation

This paper describes a technique for modeling the temporal behavior of systems which are composed of asynchronous, concurrent processes. The structure of the system can be represented in a procedure-oriented modeling language, from which an expression describing the system's state transition behavior can be derived. The derived expression can be analyzed to yield the time required to make a state transition. The time is typically a random variable, whose distribution and moments can be determined. This analysis technique is presented as a promising performance modeling method. Finally, the limitations of this method are pointed out.

Key Words: Formal modeling; message sequence; message transfer expression; modeling language; MTE; performance evaluation; PPML; system analysis.

1. INTRODUCTION

This paper is concerned with analyzing complex software systems. By complex we mean a system which is composed of non-trivial, interacting sub-parts which operate with either real or apparent concurrency. In order to analyze a system, in general, a model of some sort is required. This paper describes a class of models which can be used for system analysis. The intent of the model often determines its form. For example, one might use a Petri net to model the logical behavior of a system, while one might use a queueing network to model the performance (temporal behavior) of the system. Here we will describe a class of models which can represent both the logical and temporal behavior of a system.

The modeling technique proposed here is based on message transfer modeling developed by Riddle [1].[1] This modeling technique consists of two separate modeling languages: Program Process Modeling Language (PPML) and Message Transfer Expressions (MTE). One of the most useful properties of these two modeling languages is that an MTE model can be derived from a PPML model. A PPML model provides a procedural description of a system while an MTE model yields a more abstract, behavioral description.

Here, we are primarily concerned with MTE models and the information which can be obtained from them. An MTE, as its name implies, is an expression which describes the message transmissions

[1] References appear at the end of the paper.

which occur throughout the system being modeled. The message transfer expression indicates all possible sequences of messages which can be transmitted throughout the system as the system's state changes from a beginning state to an ending state. This sequence information is useful to determine some of the logical properties of the system, like whether or not a deadlock can occur. The MTE can also provide information about how much time is required between any two messages in a sequence. From this information the total time required for a particular sequence, or set of sequences, can be determined. Thus, using MTEs, we can determine how long it will take a system to change from one state to another.

This paper will describe the two modeling languages briefly, describe the analysis techniques available with MTEs, and discuss the limitations of this approach to system analysis.

2. PPML and MTEs

2.1 The Program Process Modeling Language

The Program Process Modeling Language, or PPML, was first introduced by Riddle [1] as a modeling method for systems which are composed of independent tasks which operate potentially in parallel and communicate with one another in a uniform manner. The language provides statement types to explicitly describe interprocess interaction (message transmission) but offers only abstract statement types to describe the internal behavior of a process (message generation). These abstractions distinguish PPML from a programming language.

The Program Process Modeling Language assumes that the system which is being described is made up of sequential processes, each being executed concurrently with, and independently of the others. These processes may communicate with one another by sending and receiving messages. The communication paths for messages being transmitted between program processes are defined by special processes called link processes, to which messages are sent and from which messages are received. Thus, the system consists of two types of processes: program processes, which generate, send and receive messages, and link processes which transmit messages.

In this view of a system all processes are asynchronous and independent of all others. Asynchrony implies that the set of processes which are logically ready to proceed may be executed in any order with any subset executing in parallel at any speed. This assumption implies a simple class of schedulers for those systems in which the number of actual processors is arbitrary. Under PPML modeling any scheduling algorithm which allocates processors to ready processes one at a time is sufficient, so long as no ready process is indefinitely excluded from execution.

All processes are independent of all other processes. The only allowed interaction among processes is communication by message transmission. This restriction implies that processes may not share memory, in the sense of modifiable data. It also implies that one process may not affect the program counter of another process. The independence requirement is necessary because the goal of PPML modeling is to represent explicitly all inter-process interaction. Since the explicit description mechanism deals with message transmission, no other interaction can be represented.

Finally, program processes communicate with other program processes by means of SEND and RECEIVE operations. Since all message transmission is accomplished by means of link processes, these operations are directed at link processes. Each program process has a "message buffer" which is part of its addressable memory. This location is the implicit source and sink for all SEND and RECEIVE operations performed by the program process. A SEND operation transfers the contents of the memory buffer to the link process named in the SEND operation. A RECEIVE operation directs a receive request at the link process named in the RECEIVE operation. When a program process performs a RECEIVE operation, it becomes logically blocked and cannot become ready again until the receive request is satisfied. A link process matches messages with receive requests. This association can be made according to an algorithm internal to the link process. The link process, in the course of a match operation, transfers the message to the receiving process' message buffer and re-enables the receiving process.

The statement types of PPML are as follows:

SET <message type> <time parameter>

> This causes the specified message type to be generated, taking the specified amount of time. Note that the time parameter may be a random variable.

SEND <link identifier>

> This causes the last message generated to be transmitted.

RECEIVE <link identifier>

> This causes the process to wait until the specified link process transmits a message to it.

IF <branch parameter> THEN
GO TO <label>

> This causes a conditional branch. The branch parameter is a binary valued random variable of known distribution.

UNLESS <message-type> GO TO <label>

> This causes a branch to <label> if the last received message is not of the specified type.

GO TO <label>

> This is the usual unconditional branch. (A more elegant, GO TO-less version of PPML is described in [2].)

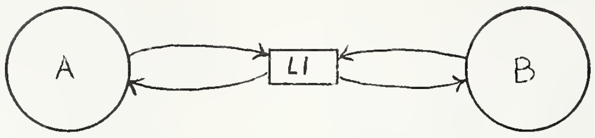
An example of a PPML model of a 2-process, 1-link system is given in figure 1.

2.2 Message Transfer Expressions

A Message Transfer Expression, or MTE, describes a set of message sequences. Each message transmission sequence can be generated by sequential or concurrent system activity. The MTE itself is quite similar to a regular expression, and can be derived from a PPML model in somewhat the same way in which regular expressions are derived from finite state automata. We will briefly define MTEs here. For the interested reader, a sizable quantity of literature concerning MTEs has appeared. A comprehensive bibliography appears in [2].

Message transfer expressions represent possible sequences of events, where the transmission of a message from a program process to a link process or from a link process to a program process is considered an event. The only identification of an event is the message type of the transmitted message. Its origin and destination are not represented. (Note: a message type represents an equivalence class of messages. Messages are identified here by their type only). Since there are usually several possible message sequences which characterize the transition of the system from a beginning state to an ending state, an MTE describes a set of sequences. As such, an MTE can be considered to define a language over a vocabulary of message types.

In addition to message types, the vocabulary of MTEs includes special symbols called synchronization symbols. The purpose of synchronization symbols is to enforce sequencing constraints on



```
         Program Process A                    Program Process B

(1)  q1:   RECEIVE L1;                 q1:   RECEIVE L1;

(2)        IF <p1> THEN GO TO q2             IF <p2> THEN GO TO q2;

(3)          SET M <t1>;                       SET M <t3>;

(4)          GO TO q3;                         GO TO q3;

(5)  q2:   SET M <t2>;                 q2:   SET M <t4>;

(6)  q3:   SEND L1;                    q3:   SEND L1;

(7)        GO TO q1;                         GO TO q1;

(8)        END;                              END;
```

Figure 1. Two Mutually Exclusive Processes

the set of message sequences by disallowing any message sequences which would violate the given constraints. Thus, the languages formed by MTEs are "cancellation" languages. The definition of message transfer expressions is:

Let

$M = \{m \mid m \text{ is a message type}\}$

$T = \{t \mid t \text{ is a time or branch parameter}\}$

$S = \{@_i \mid i=1,\ldots,n\}$

$S' = \{\overline{@}_i \mid i=1,\ldots,n\}$

where $|S| = |S'|$

$V = M \cup T \cup S \cup S' \cup \{\lambda, \emptyset\}$

where λ = the null sequence

$\emptyset$ = the empty set of sequences

A message transfer expression is defined over V as:

1. any element of V is an MTE.
2. if A and B are MTEs, then so are:

(AxB)	- sequence is either A or B
(AB)	- concatenation
$(A)^*$	- sequence is λ x A x AA x AAA x ...
$(A \Delta B)$	- sequence is A and B concurrently
$(A)^+$	- sequence is: λ x A x $A\Delta A$ x $A\Delta A\Delta A$ x ...

The Δ (shuffle) and $^+$ (concurrent closure) operators are the only additions over the operators of regular expressions. For behavioral analysis, the shuffle operator is unnecessary, since an equivalent expression can be written using concatenation and alternative (x). For performance analysis, however, this is not the case. The shuffle operator represents concurrent operation which cannot be expressed as a combination of other operators.

The synchronization symbols ($@_i$, $\overline{@}_i$) are used to require certain parts of concurrent subsequences to occur at the same time. For example, A $@$ B $\triangle$ C $\overline{@}$ D is equivalent to (describes the same set of sequences as) (A $\triangle$ C) (B $\triangle$ D).

MTEs have several interesting properties. It has been shown [2] that the class of languages which can be defined by MTEs is equal to the class of recursively enumerable languages.[2] It has also been shown [2] that, without the dagger ($\dagger$) operator, the class of languages which can be defined is equal to the class of regular languages. Without the dagger operator, the presence or absence of synchronization symbols (and the interpretation rule) makes no difference in the power of language definition.

One of the most interesting properties of MTEs is the relationship of MTEs to PPML systems. It has been shown [1,3] that message transfer expressions may be derived for any system of PPML processes subject to just one restriction on link matching algorithms. The method described in [1] for deriving an MTE from a PPML system is similar to the Brzozowski [4] derivative method of deriving regular expressions from finite state automata. This method, while theoretically workable, is quite difficult for complicated systems. Another method of MTE derivation has been suggested in [2]. This one generates an MTE quite easily, but the MTE it generates is extremely complicated.

---

[2] In [2], the terminology is slightly different from that used here. There, a distinction is made between an event expression (what we call an MTE) and an MTE (an event expression derived from a PPML model). Here, we use MTE for both cases.

The restriction on link matching algorithms is the following: a link may match any receive request at the link with any message at the link. A link satisfying this restriction is called a "bag link", because both the receive requests and the messages are unordered.

The existence of time and branch parameters in MTEs allow the time required by a sequence to be determined. Because branch parameters are random variables, and time parameters may be random variables, the time required by the set of sequences described by an MTE is a random variable. The distribution, expected value, and variance of this random variable are the quantities we are interested in.

Time parameters, as the name implies, specify the length of time required before the next message in the sequence is transmitted (its generation time). If the total length of time required by the whole sequence is what is of interest, the position of the time parameter within the sequence is inconsequential. That is, AtB = tAB = ABt. A branch parameter specifies the probability with which its associated member of a choice of sequences will occur. It makes no difference where in a concatenated sequence a branch parameter appears, e.g., pAB = ApB = ABp. Since branch parameters specify the probability with which one of two alternatives is taken, they generally appear in pairs, p and q = 1-p. This is how they are generated from PPML though subsequent manipulation may combine or eliminate some branch parameters.

As an example, the following MTE:

$$\begin{array}{l}(M((p_1\ t_2 \times q_1\ t_1) \times \\ \quad (p_2\ t_4 \times q_2\ t_3))M)^*\end{array}$$

corresponds to the PPML model in figure 1, when:

initial state is
link 1 contains M and both process A and B are waiting for a message from link 1

final state is
same as initial state

3. MTE Evaluation

3.1 MTE Manipulation

The time and branch parameters of an MTE may be combined in straightforward ways. The following axioms govern the combination of these parameters:

(a,b,c,d are time parameters, p,q=1-p are branch parameters, A,B,C,D are message types)

E.1 $aAbB = (a+b)\ A\ B$

E.2 $aA \vartriangle bB = \mathrm{con}(a,b)(A \vartriangle B)$

E.3 $(pA)^{*} = q\lambda \times pqA \times p^{2}qAA \times \ldots$

E.4 $p_1(p_2A \times q_2B) \times q_1C = p_1p_2A \times p_1q_2B \times q_1C$

E.5 $A \vartriangle (pB \times qC) = p(A \vartriangle B) \times q(A \vartriangle C)$

E.6 $aA\ @\ bB \vartriangle cC\ \overline{@}\ dD = (aA \vartriangle cC)(bB \vartriangle dD)$

$= \mathrm{con}(a,c)(A \vartriangle C)\ \mathrm{con}(b,d)(B \vartriangle D)$

$= (\mathrm{con}(a,c)+\mathrm{con}(b,d))\ (A \vartriangle C)(B \vartriangle D)$

E.7 $(p(aA \vartriangle bB))^{*} = (p\ \mathrm{con}(a,b)(A \vartriangle B))^{*}$

$= (p\ aA\ @\ )^{*} \vartriangle (bB\ \overline{@}\ )^{*}$

With these axioms, the intent of the performance parameters becomes evident. The various time parameters of a concatenated sequence of messages may all be added up to give the time required by the entire sequence. The time parameters of two message sequences which are shuffled cannot simply be added due to the concurrent nature of the operator. Instead, they are the arguments to a function (which we will call "con") whose value is the total amount of time required for the shuffled sequence. The con function will be discussed later. The star operator indicates indefinite repetition. A branch parameter associated with a message sequence which is starred is intended to indicate the probability with which the sequence will be repeated at every iteration. Branch parameters must always be a probability measure, that is, the sum of the branch probabilities of each alternative sequence in the MTE must be one.

3.2 MTE Evaluation

An MTE is a description of the sequence of events, and their relationship with one another. As such, the evaluation of an MTE is very much like a classical job-shop scheduling problem. The question which is being addressed is, "how long will this set of events (job) take to finish?" This problem has been addressed in that context, though usually not in the generality presented by MTEs. Job shop modeling is usually concerned only with sequenced and concurrent activities, and not those with alternatives.

The problem as presented by MTEs is nearly the same as that addressed in [5]. In that work, a graphical representation of such activities was presented which is sufficient to describe the sequences of MTEs. Along with the graphical representation was a method for reducing the graph to obtain the mean time required to traverse the graph from beginning to ending state. The results obtained are only for mean time where the time parameters are given deterministically.

In a later paper by Beizer, Elmaghraby's results were extended to include determination of variance for networks which do not contain concurrency [6]. These results were reported in [7], and are the same as the ones which will be presented here for concatenated, iterated, and alternative sequences.

In Elmaghraby's paper, concurrent elements of the network were assumed to take place strictly in parallel. Since this is generally an unreasonable assumption for systems of program processes, we will attempt to deal with the problem of concurrent activities which are subject to scheduling on a given number of processors. There are some results from job-shop scheduling theory which apply to this part of the problem which we will make use of. Finally, we will take synchronization symbols into consideration. Synchronization symbols, which represent sequencing constraints on otherwise concurrent activities, are not unique to MTEs, but their presence greatly complicates the analysis of the classical job-shop scheduling problem.

For evaluating MTEs, the time parameters will be taken to be random variables with finite first and second moments. It is meaningless to talk about time parameters of a sequence of events with non-finite first moments. Existence of a second moment is not crucial to any of the results concerning mean time, but, obviously, if any time parameter in a sequence has a non-existent or infinite second moment, then the sequence has a non-existent or infinite variance. Time parameters must also be independent of one another. This somewhat harsh restriction is required for manipulating sums of time parameters. Time parameters must also be independent of branch parameters.

The branch parameters, as mentioned before, are binary-valued random variables. We describe them simply by their probabilities, e.g.,

$P[b=1]=p,$
$P[b=\emptyset]=q$

The sum of the branch parameters of each alternative in any set of alternatives in the MTE must be one. Branch parameters are presumed to be static. The random variable whose probability distribution function gives a branch parameter is presumed independent from other random variables in the MTE.

The following theorems about the elapsed time of an MTE can be proved. In the following, Z is the random variable corresponding to the time required by a sequence described by the MTE. $F_Z(z)$ is the probability distribution function of Z and $f_z(z)$ is the probability density function. For brevity, we will omit writing the message type elements of the MTEs, keeping only the time and branch parameters.

<u>Theorem 1</u>

For the MTE (ab) the following are true:

$$F_Z(z) = \int_0^{\infty} \int_0^{z-y} f_{ab}(x,y)\,dx\,dy \qquad (1)$$

$$f_Z(z) = \int_0^{z} f_a(x) f_b(z-x)\,dx$$

$$= f_a(z) * f_b(z) \qquad (2)$$

If a and b are independent:

$$E[z] = E[a]+E[b] \qquad (3)$$

$$\mathrm{Var}[z] = \mathrm{Var}[a]+\mathrm{Var}[b] \qquad (4)$$

Proof:

$$z = a+b \quad \text{from E.1} \qquad (5)$$

From probability theory, the theorem results are properties of a random variable which is the sum of two random variables.

Theorem 2

For the MTE pa x qb where a and b are independent and q = 1-p, the following are true:

$$F_z(a) = pF_a(z) + qF_b(z) \qquad (6)$$

$$f_z(z) = pf_a(z) + qf_b(z) \qquad (7)$$

$$E[z] = pE[a] + qE[b] \qquad (8)$$

$$\begin{aligned} Var[z] &= pE[a^2] + qE[b^2] - E[z]^2 \\ &= p(Var[a] + E[a]^2) + q(Var[b] + E[b]^2) - E[z]^2 \qquad (9) \end{aligned}$$

Proof:

Let I be the binary valued branch parameter

$$\begin{aligned} P[z \leq Z] &= P[z \leq a | I=1]P[I=1] + P[z \leq b | I=\emptyset]P[I=\emptyset] \\ &= pF_a(z) + qF_b(z) \quad (1\emptyset) \end{aligned}$$

by definition,

$$\begin{aligned} f_z(z) &= F_z'(z) \\ &= pf_a(z) + qf_b(z) \qquad (11) \end{aligned}$$

$$\begin{aligned} E[z] &= \int_z z(pf_a(z) + qf_b(z))dz \\ &= pE[a] + qE[b] \qquad (12) \end{aligned}$$

$$\begin{aligned} E[z^2] &= \int_z z^2(pf_a(z) + qf_b(z))dz \\ &= pE[a^2] + qE[b^2] \qquad (13) \end{aligned}$$

$$\begin{aligned} Var[z] &= E[z^2] - E[z]^2 \\ &= pE[a^2] + qE[b^2] - E[z]^2 \qquad (14) \end{aligned}$$

Theorem 3

For the MTE $(pa)^*$ the following are true:

$$F_z(z) = \int_0^z f_z(x)dx \qquad (15)$$

$$f_z(z) = L^{-1}\left[\frac{q}{1-pf_a{}^*(s)}\right] \qquad (16)$$

where $L^{-1}[x]$ denotes the inverse Laplace transform of x and $f_a*(s)$ denotes the Laplace transform of $f_a(z)$

$$E[z] = \left(\frac{p}{q}\right) E[a] \qquad (17)$$

$$Var[z] = \frac{p}{q} Var[a] + \frac{p}{q^2} E[a]^2 \qquad (18)$$

Under the assumptions that:

1. p is stationary, independent of all a
2. all a are independent, identically distributed

Proof:

$$Z = \sum_{i=\emptyset}^{N} a_i \qquad (19)$$

where $a_i = a$ are i.i.d. N is a random variable with density $f_N(i) = p^i q$

$$f_z(z) = \sum_{i=\emptyset}^{\infty} [f_a{}^i(z)]p^i q \qquad (2\emptyset)$$

where $f_a{}^i(z)$ is the i-fold convolution of $f_a(z)$

$$f_z^*(s) = \sum_{i=0}^{\infty} [f_a^*(s)]^i p^i q$$

$$= q \frac{1}{1-pf_a^*(s)} \qquad (21)$$

so

$$f_z(z) = L^{-1}\left[\frac{q}{1-pf_a^*(s)}\right] \qquad (22)$$

More generally, note that

$$f_z^*(s) = \sum_{i=0}^{\infty} [f_a^*(s)]^i P[N=i]$$

defines the Z-transform of N,

$$\text{so, } f_z^*(s) = N(f_a^*(s)) \qquad (23)$$

$$E[N] = q \sum_{n=0}^{\infty} np^n \qquad (24)$$

$$E[N^2] = q \sum_{n=0}^{\infty} n^2 p^n$$

$$= \frac{p(1+p)}{q^2} \qquad (25)$$

$$Var[N] = \frac{p(1+p)}{q^2} - \frac{p^2}{q^2} = \frac{P}{q^2} \qquad (26)$$

$$E[Z] = E[E[a|N]] = E[N]E[a]$$

$$= (p/q)E[a] \qquad (27)$$

$$Var[Z] = E[Var[A|N]] + Var[E[Z|N]]$$

$$= E[N]Var[a] + E[E[Z|N]^2] - E[E[Z|N]]^2$$

$$= E[N]Var[a] + E[N^2E[a]^2] - E[N]^2E[a]^2$$

$$= E[N]Var[a] + E[N^2]E[a]^2 - E[N]^2E[a]^2$$

$$= E[N]Var[a] + E[a]^2(E[N^2] - E[N]^2)$$

$$= E[N]Var[a] + E[a]^2Var[N]$$

$$= \frac{p}{q} Var[a] + \frac{p}{q^2}E[a]^2 \qquad (28)$$

These theorems provide the tools for evaluating an MTE without concurrency. See figure 2 for an example MTE evaluation.

Obviously, no such theorem can be given for concurrent MTEs in the absence of specification of the scheduling algorithm to be used. The number of processors in the system will clearly affect the length of time required for the completion of the concurrent sequences. Total required processing time, of course, is determined just as in the case of concatenated sequences -- it is merely the sum of the time parameters. The more interesting problem of determining the required real time for the behavior represented by the MTE is more difficult. To determine the actual time the sequence will take requires more information than is contained in the MTE. That is, the number of processors available in the system and the scheduling algorithm must both be known. In addition, if the scheduling algorithm is not sufficiently simple, a closed form solution may not be obtainable.

Previously, time parameters for a shuffled (concurrent) sequence were combined by a "con" function, i.e., $t_1A \Delta t_2B = con(t_1,t_2)(A \Delta B)$. The value of the function depends on the number of processors available. Each parameter represents a required amount of processing time which can proceed concurrently with the processing represented by all other parameters. Two special cases are readily apparent. If there is but a single processor, the time required to complete all the

$p\ (p_1t_2 \times q_1t_1) \times q\ (p_2t_4 \times q_2t_3)$

where $p = 1 - q$ is the probability assiciated with the link matching a message M with a receive request from process A. This probability will usually be 1/(number of possible receives) = 1/2.

$$E[Z] = p\ (p_1E[t_2] + q_1E[t_1]) + q\ (p_2E[t_4] + q_2E[t_3])$$

$$Var[Z] = E[Z^2] - E[Z]$$

$$E[Z^2] = p\ (p_1E[t_2^2] + q_1E[t_1^2]) + q\ (p_2E[t_4^2] + q_2E[t_3^2])$$

Figure 2. Evaluating the MTE Corresponding to Figure 1

processing is the sum of the function parameters, regardless of the scheduling algorithm used. If there are at least as many processors as concurrent activities, then the time required to complete the sequence is simply the maximum value of the function parameters. This will be true for all scheduling algorithms which satisfy the PPML requirements for schedulers.

For the case of a sufficient number of processors, the following theorems hold:

Theorem 4

For the MTE $a \triangle b = con(a,b)$ the following are true if a and b are independent:

$$Z = con(a,b) = max(a,b) \qquad (29)$$

$$F_z(z) = F_{ab}(z,z) = F_a(z)\ F_b(z) \qquad (30)$$

$$f_z(z) = f_a(z)F_b(z) + f_b(z)F_a(z) \qquad (31)$$

Proof:

From the definitions of Z and the distribution functions F_a and F_b

Theorem 5

For the MTE $(a)^+$ the following are true:

$$Z = max\ (a_1, a_2, \ldots, a_n) \qquad n \text{ a value of } N \qquad (32)$$

$$F_z(z) = \frac{q}{1-pF_a(z)} \qquad (33)$$

$$f_z(z) = \frac{pqf_a(z)}{(1-pF_a(z))^2} \qquad (34)$$

for a_i independent, identically distributed random variables

Proof:

$$F_{zn}(z) = P[max(a_1, \ldots, a_n) \leq z] = F_a(z)^n \qquad (35)$$

so

$$F_z(z) = \sum_{n=0}^{\infty} F_{zn}(z)P[N=n] = \sum_{n=0}^{\infty} F_a(z)^n p^n q = q\left(\frac{1}{1-pF_a(z)}\right) \qquad (36)$$

$$f_z(z) = d\,F_z(z)$$

$$= q \sum_{n=0}^{\infty} (f_a(z)\; n\, F_a(z)^{n-1}) p^n$$

$$= q\, f_a(z)\; p \sum_{n=1}^{\infty} n\, F_a(z)^{n-1} p_{n-1}$$

$$= \frac{pq\, f_a(z)}{(1-pF_a(z))^2} \qquad (37)$$

For cases other than the two special cases (1 < number of processors < number of processes) there are no general results, unless the scheduling algorithm is given. Most scheduling algorithms are too complicated for closed form results, but a useful result to obtain in many cases is the minimum possible time required. For this, there are usable results from job-shop scheduling theory. In fact, McNaughton's formula provides the minimum time required to complete a set of concurrent tasks, given a set of processors.

<u>McNaughton's formula</u>

For m processors and n concurrent processes,

$$\min \mathrm{con}(t_1, t_2, \ldots, t_n) = \max(1/m \sum_j t_j, \max_j(t_j)) \qquad (38)$$

in the case where any processor may be preempted from any process at any time and process switching time is negligible.

By preemption it is meant that any process can be interrupted and resumed later without losing the work done before the interruption[3]. The requirement that preemption be allowed for the application of McNaughton's formula is satisfied in PPML systems by the independence requirement of concurrent processes. By definition, shuffled sequences are produced by concurrent processes and can, therefore, proceed in any order, in any number of sub-operations at a time.

There is an algorithm for scheduling processors to processes which will realize the optimum time. Unfortunately, it requires a prior knowledge of the time required by each sequence. In multiprocessor computing systems, the scheduling algorithm usually implements some sort of round-robin scheduling where each process gets a little bit of processor time every so often. In the case where each process gets an equal amount of processor time at a time (a time slice), the time to complete all the processes (complete the sequence of the MTE) approaches the minimum as the value of the time slice is reduced. The minimum would, therefore, be reached when each processor would execute a single instruction from each process in turn. Unfortunately, in doing this, the assumption of negligible process switching time is usually violated.

Finally, the effects of synchronization symbols must be analyzed. Synchronization symbols provide sequencing constraints on concurrent activities. That is, they force some part of a sequence to occur after some part of another otherwise concurrent sequence. Consequently, the following lemma holds.

<u>Lemma 1</u> For the MTE

$$(t_1\, A \,@\, t_2\, B) \,\Delta\, (t_3 C \,\overline{@}\, t_4 D)$$

the following is true:

$$E[Z] = E[\mathrm{con}(t_1, t_3) + \mathrm{con}(t_2, t_4)]$$
$$= E[\mathrm{con}(t_1, t_3)] + E[\mathrm{con}(t_2, t_4)] \qquad (39)$$

$$\mathrm{Var}[Z] = \mathrm{Var}[\mathrm{con}(t_1, t_3)] + \mathrm{Var}[\mathrm{con}(t_2, t_4)] \qquad (40)$$

---

3 If preemption is not allowed, the equality of McNaughton's formula becomes greater than or equals.

Proof:

The proof follows from the definition of synchronization symbols.

Synchronization symbols often arise in MTEs to express a relationship between two sequences which are each repeated indefinitely. In the sequence $(p\ @\ a\ A)^* \Delta\ (\bar{@}\ b\ B)^+$, the intent of the @ symbol is to relate the number of B sequences to the number of A sequences. The dagger operation, in addition, allows the B sequences to proceed concurrently, once they begin. Graphically, this situation would look like figure 3.

Analyzing such a sequence is fairly difficult. The required time again depends on the number of processors and the scheduling algorithm. The minimum required time can be obtained, however, by theorem 3. That is, the time required to produce the iterated sequence of A's is clearly the minimum time of the sequence. This minimum will only be obtained when there are a sufficient number of processors available to allow the A sequence to proceed continuously and to complete each B event before a new B event for which there is no available processor occurs. Needless to say, this is not a very general result. If the A sequence is a short one in comparison to the B sequences, then an approximation to the required time is given by con$(nTA, TB_1, \ldots, TB_n)$ where n = the number of iterations of the A-loop. It appears that very little of a general nature can be said about indefinite sequences containing synchronization symbols.

4. System Analysis Using MTEs

The only performance result directly available from evaluating an MTE is time required to complete the sequence. In theory, at least, an MTE is available for any beginning and ending state pair of a system. Thus, the time required for a system to change from one state to another can be obtained. Most of the standard performance measures are derivable from the

```
    a       a        a       a        a        a
+-------+--------+--------+--------+--------+---------  etc.

     b
+----------

            b
        +-----------

                     b
                +------------------

                         b
                         +-----------

                                  b
                                 +------------
```

Figure 3. Possible Timing for the Sequence $(p\ @\ a\ A)^* \Delta\ (\bar{@}\ bB)^+$

state transition time: for instance, response time, service time, throughput. It must be admitted that obtaining these measures is not especially easy using MTEs. In fact, for a measure like queue length at a congestion point (a link process), MTE analysis gives very little help.

The response time of a system is apparently well suited to MTE analysis because response time is merely the time elapsed between two messages -- a START message and a DONE message. This value is provided directly by evaluating the MTE. However, the context of the PPML system and the MTE may complicate its determination. The main requirement for producing an MTE, given a PPML system satisfying the appropriate restrictions, is the specification of a beginning and ending state. These two states must be given explicitly for the MTE to be derived. In many applications, it is not realistic to completely specify the beginning and ending states. For instance, in the case of a system with several program processes which represent users, the desired result is the expected response time for any one of the user processes making no assumptions about the states of the other processes. The beginning and ending states of interest would be the partial states giving the process of interest's START and DONE messages and other processes in arbitrary states. From such state descriptions, MTEs cannot be derived. The best that could be done is to specify all possible states of each other process as an alternative in the beginning and ending states.

However, MTEs can be used to obtain interesting interevent times in a system by using other knowledge about either the events, or the system, or both. For instance, an MTE can be derived for the state "each process sends a START" and "each process received a DONE". By making appropriate assumptions about the number of processors, homogeneity of processes, etc., the response time in general may be inferred.

As an example of the use of an MTE in system analysis, consider the PPML system in figure 4. For the state transition corresponding to "both processes start" to "both processes finish" we can write the MTE as:

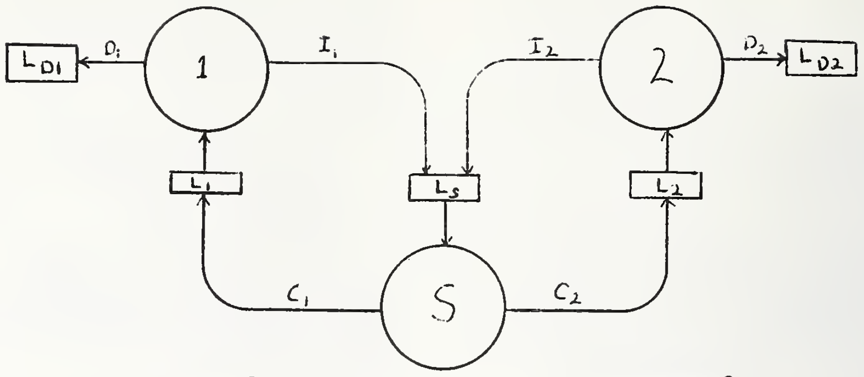
$$
\begin{array}{l}
(p_1\ a\ @_1\ b\ @_2)^*_* c\ \triangle \\
(p_2\ d\ @_3\ c\ @_4)^* f\ \triangle \\
(\bar{@}_1\ \bar{@}_2\ x\ \bar{@}_3\ \bar{@}_4)
\end{array}
$$

Again, we have left out the message types, since our only interest here is in evaluating the time required. Notice also that $p_i=1-q_i$.

Before evaluating this MTE, we can point out that the action of process S has become distributed into the two terms. The fact that b and c are both performed by the same process gives rise to the presence of the synchronization symbols. The third term in the MTE, consisting only of synchronization symbols, causes either b or c to occur, but not both at once. It is readily apparent that the presence of this synchronization will complicate the analysis. None of the previous theorems can handle this expression.

The measure of interest here is response time, i.e., how long will it be from the start of a process until it sends the done message to its link process. The time required by the entire MTE is not really of interest to this question, since that represents the time required for both processes to respond. In order to determine the response time, we really only need to evaluate the term corresponding to the process of interest, and the interference effects with other terms. For process 1 we just have to evaluate $(p_1\ a\ @_1\ b\ @_2)^*c$ and the interference with process 2.

Determining the interference between the two terms is a non-trivial problem. If we assume there is a single processor which is shared, the interference can be determined by simply evaluating each term and increasing the elapsed time of both terms by the shorter of the two. This, however,



```
        Process 1                          Process 2

loop:                                 loop:
        If <q1> THEN GO TO done;              If <q2> THEN GO TO done;
           SET I1 <a>;                           SET I2 <d>;
           SEND Ls;                              SEND Ls;
           RECEIVE L1;                           RECEIVE L2;
           GO TO LOOP;                           GO TO LOOP;
done:                                 done:
        SET D1 <c>;                           SET D2 <f>;
        SEND LD1;                             SEND LD2;
        END;                                  END;
```

```
                 Process S

         loop:
             RECEIVE Ls;
             UNLESS I1 GO TO p2;
             SET C1 <b>;
             SEND L1;
             GO TO LOOP;
         p2:
             SET C2 <c>;
             SEND L2;
             GO TO LOOP;
         END;
```

Figure 4. An Example PPML System

is quite a restrictive assumption. Particularly at low levels of a system, a processor sharing assumption is often not appropriate.

If we assume a multiprocessor environment, things get more complicated yet. In fact, this system could be represented by a multiple-class, closed queueing network. By assuming exponential (or Erlang) distributions for the time parameters, we could solve this problem by using queueing theory. Without those assumptions, there is no general solution technique.

We can make the observation here that PPML systems bear a close resemblence to closed queueing networks, a fact that will not have escaped the reader familiar with queueing networks. However, while the example can be translated into a network of queues, PPML systems, in general, include a larger class of systems. This is because processes (nodes) generate messages (transactions) both in response to messages received and spontaneously. There is not necessarily a one-to-one correspondence between messages received by a process and messages sent by it. It should be apparent that any queueing network, open or closed, can be realized by an equivalent PPML system. Therefore, PPML systems, and MTE systems, are a superset of systems representable by queueing networks.

5. Limitations of MTE Analysis

As has been shown, formal analysis of a PPML system by MTEs consists of three parts: deriving the MTE, evaluating the MTE, and relating MTE results to the operation of the system. Each of these three steps imposes restrictions on the kinds of systems that can be analyzed. In the derivation process, several substantial restrictions must be imposed.

The most important restriction placed on PPML systems for deriving MTEs is the requirement that all links use a random matching algorithm (that is, each link is a bag). The consequences of this requirement with respect to the behavior of analyzable PPML systems is a subject of current research (see [2]). It is obvious, though, that such a restriction may require awkward or inelegant system models for many systems, even though those systems' behaviors are still realizable under the bag restriction.

Another important limitation of the derivation process is the requirement that beginning and ending states be completely specified. The MTE simply cannot be derived if the two states are not completely specified, which is often impractical.

The final restriction on the derivation process is the complexity of the process itself. It is extremely difficult to derive MTEs and manipulate them into useful forms. This situation is not hopeless, however, as research on MTE derivation continues. New derivation techniques may be found which will work for more and more PPML systems.

In the evaluation phase of MTE analysis, most of the limitations are imposed by the probability arguments of the theorems. The main requirement is independence of the random variables which represent the performance parameters. Depending on the level of description of the PPML system, this may or may not be an unrealistic assumption. At high levels, independence is often a realistic assumption because the various performance parameters represent operations which are fairly disjoint. At lower levels, the performance parameters become less independent.

For theorem 3, the looping theorem, to hold, the time parameters must be stationary and independent of one another. That is, the time required going through the loop the first time cannot affect the time required to go through the loop subsequent times. This is not an unrealistic assumption, but the theorem also requires that branching probabilities be stationary. This means that the probability of exiting the loop does not change with the number of times the loop is repeated. In many cases this is an unrealistic assumption.

For determining the time required for concurrent sequences, few tools are available. Although McNaughton's formula gives the minimum possible time for a set of concurrent sequences, there are no results which can be used to determine the actual required time given the scheduling algorithm. This is precisely the question most in need of answer for the evaluation of many systems.

Likewise, there are few results available for MTEs containing synchronization symbols. If the MTE contains simple synchronization, lemma 1 may be applied, but in the more complex cases little help is available from the evaluation rules. As we saw in section 4, simple synchronization can lead to complex evaluation problems.

Finally, the application of MTE results to system evaluation is not direct. Care must be used in determining the beginning and ending states of interest and ingenuity often must be applied in relating the results to the standard measures of performance. Again, there is no method for determining any of the standard performance measures for all systems.

6. Conclusions

The PPML and MTE modeling techniques show considerable promise for both the behavioral and temporal analysis of a system. The method, however, still requires a good deal of development before it becomes a useful tool. The main areas requiring work are the derivation of MTEs from PPML models, which was not addressed in this paper, and the evaluation of MTEs. We presented several results for evaluating MTEs, but we have also seen the limitations of these results when applied to real MTEs generated from PPML models. However, it is encouraging to note the similarities between MTE evaluation and other system evaluation techniques, particularly job-shop scheduling and queueing network analysis. Results from these other disciplines can be used for MTEs.

7. References

1. Riddle, W.E., _The Modeling and Analysis of Supervisory Systems_. Ph.D. Thesis, Stanford University, Computer Science Department, March 1972.

2. Riddle, W.E., _An Approach to Software System Modeling, Behavior Specification and Analysis_. RSSM/25 University of Michigan, Department of Computer and Communication Sciences, July 1976. (To appear in the Journal of Computer Languages.)

3. Sanguinetti, J.W., _Performance Prediction in an Operating System Design Methodology_, Ph.D. Thesis, University of Michigan, Department of Computer and Communication Sciences, 1977.

4. Brzozowski, J.A., Derivatives of Regular Expressions, _Journal of the ACM_, Vol. 11, No.4, Oct. 1964.

5. Elmaghraby, S.E., An Algebra for the Analysis of Generalized Activity Networks, _Management Science_, Vol. 10, No. 3, April 1964.

6. Beizer, B., Analytical Techniques for the Statistical Evaluation of Programming Time, _AFIPS Conference Proceedings_, Vol. 37, Fall 1970.

7. Graham, R.M., Performance Prediction in Software Engineering in _Lecture Notes in Computer Science_, 30, Springer-Verlaag, 1975.

PERFORMANCE EVALUATION WITH PETRI NETS

Y. W. Han

Bell Telephone Laboratories
Naperville, Illinois 60540

Petri net-like models depict the concurrently of a system precisely and concisely, and are especially suitable for modeling distributed data processing systems. After expressing a system by a Petri net-like model, this paper suggests and discusses methods to: (1) Identify system bottlenecks, (2) Derive the maximum subsystem utilization of every subsystem, (3) Define quantitatively a measure for the cost-effectiveness of a system, (4) Formulate the "peakload" of a system, and (5) Calculate the waiting time, maximum queue length, and average queue length of every queue in a system.

Every transition (subsystem) in this study takes a fixed amount of time to fire (activate), and only closed system are considered. If the activation time of a subsystem is not fixed, then the maximum, the minimum, or the mean activation time can be used to provide insights into system performance.

Key words: Concurrency; bottlenecks; overload; peakload; waiting time; queue length.

1. Introduction

Evaluating performance of a digital system is difficult because of the complexity of the system, interfaces among subsystems, interaction between hardware and software, and changing system requirements and workloads. According to the stages of a system and the availability of data, simulation, analysis, and measurement are used to predict system performance, to identify bottlenecks, to justify design decisions, and thus to help derive design alternatives. To facilitate performance evaluation, various models have been widely used for either simulation or analysis, such as queueing models, directed graphs, and simulation languages. Recently, Petri net-like models have been used for both simulation and analysis,[1] as well as for verifying designs.[2]

A complete and flexible Petri net-like model will be discussed. It is able to represent all systems precisely and concisely. One unique usage of this model is to support a widely accepted top-down, hierarchical design methodology. The spirit of this methodology is to decompose the development of a large system as multiple layers, namely, requirements specifications, system architectural design, subsystem design, coding, laboratory testing, field testing, installation, system updating, etc. Basically, the process is hierarchical and a layer starts only if all previous layers have been completed or are well understood, and only if they are all consistent. The Petri net-like model can be used as a specification tool to specify the system hierarchically. To verify all available layers are consistent, system designers have to predict the performance of a layer. After modeling a system at a layer, this paper provides an engineering, analytic approach to predict system performance. In case that certain results of a system are hard to obtain analytically, it is always possible to get their results by simulating the system which is specified by the Petri net-like model. The combination of simulation and analysis is a powerful design and evaluation tool.

A Petri net is a directed graph with two types of nodes. Conditions are represented by circles (◯) called places, and events are represented by bars (—) called transitions. Arrows which connect transitions to places or vice versa are called links, edges, or arcs. The firing rule for a Petri net is illustrated in Figure 1. Places p_1 and p_2 are the input places of transition 1, and places p_3 and p_4 are the output places. In turn, t_1 is the output transition of p_1 and p_2, and is the input transition of p_3 and p_4. Whenever every input place of a transition contains at least one token, the transition is enabled to fire. The firing of a transition removes a token from each of its input places and adds a token to each of its output places. Figure 1(a) shows the marking of tokens before transition 1 fires, and Figure 1(b) shows the marking after transition 1 fires.

It has been shown that Petri nets with negation[7], illustrated later in Figure 2(b), have the same representation power of Turing machines, which are able to represent all systems. We shall show that Petri nets with imposed priorities on enabled transitions can represent negation and therefore can represent all systems. Figure 2(a) depicts a Petri net in which place p inhibits transition t_b if transition t_a has priority over t_b. This is equivalent to Figure 2(b) in terms of the relation between place p and t_b, which is a negation. Thus, Petri nets with priorities can represent all systems, and are a complete model. Furthermore, the conciseness of these nets make them applicable to real system evaluation.

We have proposed a model which has a control graph and a data graph.[3,4] Control graphs are Petri nets with priorities and show precedence relationships. Data graphs show data storage, transformation, and testing. In a control graph, if a place has two or more output transitions, then the transition that is fired by a token in the place depends on the testing result in the corresponding data graph. In this paper, only control graphs are considered. Since the distinctions between control graphs and Petri nets do not affect the forthcoming analysis, these terms are sometimes used interchangeably for simplicity.

A Petri net itself is an uninterpreted model. By adjusting to different applications, a system designer or evaluator can flexibly interpret a token, a transition, a place, and the firing of a transition to fit the current needs. Through the motion of tokens, Petri nets or Petri net-like models are suitable for simulating discrete events. As queueing models, they can also indicate system behaviors in equilibrium. In fact, queueing theories should be extended to Petri net-like models to overcome some of the drawbacks of current queueing models, such as:

- Nonequilibrium status is not considered practically. In real-time systems there are stringent time requirements on tasks. A real-time system is said to be in an overload condition with a given workload when certain time requirements cannot be fulfilled. An overload strategy is the mechanism that coordinates the system activities to resolve overload conditions after an overload has been detected. Overload strategies are certainly a central issue for determining the system capacity and are of major importance. But we cannot use queueing theories to evaluate an overload condition because systems are not in equilibrium.

- A user (task) can occupy at most one service station (hardware device). Nevertheless, in a multiprocessor environment, several processors may simultaneously work on a task, e.g., Illiac IV.[5]

- Every task is independent of the other tasks in a queueing network. As a result, some significant events in a digital system network are not representable by queueing networks, such as the updating of multiple-copy data distributed in different locations of a network.

These drawbacks can be alleviated by extending queueing theories to Petri net-like models or by using Petri net-like simulation.

To relate Petri net-like models with queueing networks, we interpret the physical meaning of transitions, places, and tokens as follows:

Component	Physical Meaning
Transition	Service station (subsystem)
Place	Queue
Token	Task (or customer)
The number of tokens in a place	Queue length
Transition firing time	Service time

This paper discusses some analytic results of using a Petri net-like model for performance evaluation. The frequency of activating a service station, the identification of system bottlenecks, and resource utilization are presented in Section 2. The methods of calculating waiting time, maximum queue length, and average queue length for every queue of a system are described and illustrated in Section 3. To find the average queue length we convert timed nets to state diagrams. The conversion of nets or timed nets to state diagrams is a way of simulating the nets. Generalizing from the result of a Markov chain, subsequently we propose a method to decide the probability of a system's being in a state of the state diagram.

2. TRANSITION FIRING FREQUENCIES, SYSTEM BOTTLENECKS, AND RESOURCE UTILIZATION

2.1 Transition Firing Frequencies

The firing of a transition in the nets removes a token from every transition input place and adds a token to every transition output place. Thus, the conservation of tokens at a place p_k, as shown in Figure 3, requires[6].

$$\forall k \quad \sum_{i=1}^{m} f_{x_i} = \sum_{j=1}^{n} f_{y_j}$$

where f_{x_i} and f_{y_j} denote respectively the number of times the ith input transition and jth output transition fire.

Let us call the preceding equation the conservation of token law in place p_k. Let f_j denote the number of times that transition j fires relative to the other transitions. Applying the conservation of token law to every place of a net, we get a set of simultaneous equations in terms of variable f_j. If a positive, nonzero solution exists for every f_j, then this solution determines the relative transition firing frequencies. Such a net is called consistent.

This paper is concerned only with consistent nets. Readers interested in the details of the the properties of consistent and inconsistent nets are referred to Han and Kinney.[6]

2.2 System Bottlenecks

We have presented a method of determining relative transition firing frequencies. The following theorem identifies system bottlenecks. A system bottleneck is a transition which limits the throughput rate of the system. The throughput rate of a system is the average number of tasks handled by the system in a unit of time.

Theorem 1: Assume that T_i is the time to fire transition t_i. Then the bottleneck of a system with n transitions is at transition t_j, iff $f_jT_j = \max(f_1T_1, f_2T_2, \cdots, f_nT_n)$.

Proof: We shall prove it by contradiction. Suppose $f_jT_j = \max(f_1T_1, f_2T_2, \cdots, f_nT_n)$ and the system bottleneck is at t_i with $f_iT_i < f_jT_j$. Since t_i is the system bottleneck with a service time T_i, then by definition transition t_i can have an absolute frequency of $\frac{1}{T_i}$.
Transition t_j, therefore, has a firing frequency of $\frac{f_j}{f_iT_i}$, which is impossible.

The other direction of the proof can be done similarly. Q.E.D.

For instance, the bottleneck in Figure 4 is t_2, because $f_1 : f_2 : f_3 = 10:7:3$, and $f_1T_1 : f_2T_2 : f_3T_3 = 10:14:12$. The numbers between p_1 and t_2, and between p_1 and t_3 are respectively the probabilities that the tokens in p_1 will fire t_2 and t_3.

The identification of system bottlenecks is of central importance in improving system performance. If only one bottleneck exists, either parallel processing of the bottleneck or subdividing it into subunits will remove it and, using theorem 1 again, will identify the transition that is the new bottleneck in the improved system. Suppose we want to add a new transition, t_k, to a net to provide more features in the existing net. We certainly do not want $f_kT_k > f_jT_j$, or t_k becomes a new bottleneck and reduces the throughput

2.3 System Utilization, Peakload, and Cost Effectiveness

If a transition is a bottleneck, then at a peakload this transition activates (fires) continuously. In other words, the transition is utilized 100 percent. Assume that t_j is the bottleneck; then the maximum utilization of a transition, t_i, is

$$\frac{T_if_i}{T_jf_j} \times 100\%$$

The preceding formula is true because:

- Transition t_i fires T_if_i units of time, while the bottleneck t_j fires T_jf_j units of time.

- At a peakload or overload, the bottleneck fires continuously.

This formula leads us to derive the "peakload" of a system. First, let us define peakload. When the workload of a system is increased, the throughput rate and the system utilization will be increased until the workload reaches the peakload (see Figure 5). In other words, the peakload of a system is the minimum workload it takes to reach the maximum throughput rate or system utilization. However, in reality, the curve of system utilization versus the workload is only asymptotic to the one shown in Figure 5, if there are variances in service times and/or there are branches in the system. In these cases, it is almost always possible to have a small increment in system utilization by increasing the workload. Thus, the peakload of a closed system is defined to be the minimum number of tokens (tasks) which ensures the emergence of bottlenecks constantly. On the other hand, the turnaround time of a system is a monotonically nondecreasing function of the workload. For a real-time system, there are stringent time requirements. Therefore, it is of practical importance to find the peakload of a given system.

Since we already know the maximum utilization of every transition, then the peakload is the minimum number of tokens (load) it takes to make every transition reach that utilization. To activate a transition, each of its places must have at least one token. Therefore, the minimum number of tokens to make a transition busy seems equal to the number of its input places. However, a token in various places may have different weights. As an example, in Figure 7, a token in place p_1 is equal to a token in p_2 and a token in p_3. The following method specifies how to determine the weight of every place in a net.

If we assume that the weight of a token in a place p_i is w_i and that the firing of every transition conserves the total weight of a system, we then get a set of simultaneous equations in terms of w_i. If a set of positive integer solutions exists for every w_i, which is the weight of p_i, then the net is said to be invariant. Otherwise, the net is variant. This paper is concerned only with invariant nets. Invariance is a dual property of consistency. The properties of invariant nets and their relationship to consistency are documented in Han and Kinney.[6]

Figure 7 is an example of an invariant net in which $w_1=2$, $w_2=1$, and $w_3=1$. With the obtained weight of a token in every place and the derived maximum resource utilization of every transition, the peakload of a system is expressed in terms of weighted tokens as

$$\frac{\Sigma W_iT_if_i}{T_jf_j}$$

where $W_i = \sum_{j=1}^{n} w_{ij}$, and w_{ij} is the weight of a token in an input place of transition t_i, assuming that t_i has n input places. W_i, thus, is the minimum number of weighted tokens it takes to $\frac{T_if_i}{T_jf_j}$ percent of its time in firing and is idle the rest of its time. In a simple example (Figure 8), the peakload is two tokens, which are enough to maximize the utilization of all the transitions.

If c_i, the cost of using transition t_i, is known, the cost effectiveness of a system is defined as

$$\frac{\Sigma c_iT_if_i}{(\Sigma c_i)T_jf_j}$$

This is a yardstick for measuring the cost effectiveness of a design.

In this section, consistent and invariant nets have been defined. The methods of determining the existence of consistency and invariance have been specified. By determining the consistency of a net, the relative transition firing frequencies can be obtained if the net is consistent. Given the obtained relative transition firing frequencies and the transition firing times of a net, theorem 1 can specify the characteristics of system bottlenecks and thus identify them. Subsequently, the maximum utilization of a transition can be derived. If a net is invariant, the weight of a token in every place of the net can be found. From the derived maximum utilization of transitions and the weights of places, the peakload of a system is derived. Later the cost effectiveness of a system is also defined, based on the concept of the maximum utilization of a transition.

In Section 3, we shall discuss and derive the waiting time and the maximum and mean queue lengths of any queue in a system.

3. WAITING TIME, MAXIMUM QUEUE LENGTH, AND AVERAGE QUEUE LENGTH

Little[10] states that the average number of tasks (also called "customers" or "input") equals the average arrival rate of tasks to that queue times the average time a task spends in that queue. To fit our terminology, we rephrase the statement as follows: the number of tokens in a place equals the average arrival rate of the tokens times the average time a token spends in the place; that is.

$$\overline{N}_i = \lambda_i A_i$$

where $\overline{N}_i$ = the average number of tokens in place i, or the average queue length

λ_i = the arrival rate of the tokens

A_i = the average time a token stays in place; this is equal to the sum of waiting time and of service time.

With Little's result, we can derive the third variable after two out of the three variables ($\overline{N}_i$, λ_i, and A_i) are known. In the previous section, ways of determining relative transition firing frequencies and system bottlenecks are described. At a peakload, the absolute frequencies of bottlenecks are decided by their transition firing times. In other cases, the first absolute frequency needs to be measured. From one absolute firing frequency of a transition and from the relative transition firing ratios (which were previously called relative firing frequencies), the absolute transition firing frequencies are obtained. The arrival rate of a place p_i is obtained from the sum of the firing frequencies of its input transitions.

To find the maximum queue length and minimum queue length of a place in a net, we will first simulate a net as a state diagram. A state in the state diagram is identified by, or partly by, the number of tokens in each place of the net. Scanning over all the states, we easily find the maximum queue length and the maximum number of tokens of each place.

However, to obtain the average queue length, it is necessary to calculate the probability of the system's being in every state of the state diagram. We shall propose a way to calculate the probability and thus obtain the mean queue length. By using Little's result, together with the obtained mean queue length and arrival rate discussed previously, the average time a token (task) is in a station can be derived as well as the waiting time.

First, let us consider the states of a net and of a timed net.

3.1 States of a Net

If a net is in a state, then the next state depends only on the current state and is independent of the path by which the net comes to the current state. Given a net, its state is expressible as the distribution of tokens in the places. A transition can fire if one or more tokens exist in each of its input places. After firing a transition, a token is removed from each input place and added to each output place. Figure 8(b) is a state diagram representation of Figure 8(a). In Figure 8(b), a state is expressed as (n_1, n_2), where n_1 and n_2 denote respectively the number of tokens in p_1 and p_2. The details of the transformation are given in Han,[8] which is a straightforward extension of Karp and Miller's rooted tree.[9] From the diagram we can see the maximum queue length is two for both places.

Firing a transition results in the occurrence of certain actions. In reality, the firing of a transition takes time. Throughout this paper, for simplicity, we assume that the firing of a transition consumes a prespecified, fixed amount of time. When a transition is enabled and then decides to fire, one token from each of its input places is committed to the transition for that fixed amount of time. Subsequently, the tokens will be removed from the input places and one token will be added to each output place. This action of removing and adding tokens consumes no time.

Considering transition firing times, we observe that certain transition firing sequences do not occur. But for the retained firing sequences, the distribution of tokens alone is not enough to represent the states. A state of a timed net is expressed as the combination of the token distribution in the places and the times that have been spent in each transition. As an example, Figure 8(d) is the state diagram representation of Figure 8(c). In Figure 8(d), a state is denoted as (n_1,n_2) $\langle m_1,m_2,m_3\rangle$, where n_i is the number of tokens in place i, and m_j is the time spent firing transition t_j.

The number on an edge from node i to node j in Figures 8(c) and (d) is the probability that the next state will be in node j when the current state is in node i.

For a token in a place that has two or more transitions, as in place p_1 in Figure 8(c), the transition that is fired depends on a data test in the corresponding data graph.[6] For simplicity, we assume that the data test consumes a negligible amount of time in this example. Because the data test determines the paths of a token, two tokens in place p in Figure 8(c) do not guarantee that transitions t_1 and t_2 will activate simultaneously. In Figure 8(d), 0^+ is positioned in m_i to indicate that a token will fire transition t_i.

3.2 Probability of Being in a State

To calculate the average length of a queue, we need to consider the probability of the system's being in every state. In general this process is quite involved. We shall discuss the simplest case first, using the example in Figure 4. For ease of discussion, we have numbered the states in Figure 8(d) and have redrawn it as Figure 8(e). In this special example, the system stays in each state for one unit of time, coincidentally. This system also has the Markov property: the probability of being in the next state depends only on the current state.

Let us define

π_i = the probability that the system is in state i

Π = the probability vector

$\Pi = \left[\pi_1,\pi_2,\pi_3,\pi_4,\pi_5,\pi_6,\pi_7\right]$ in the example.

m_{ij} = the transition probability moving from state i to state j

M = the transition probability matrix

$M = \left[m_{ij}\right]$

In this example, shown in Figure 4(e),

$$M = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

It is certain that

$$\pi_1+\pi_2+\pi_3+\pi_4+\pi_5+\pi_6+\pi_7 = 1$$

Using the property of a Markov Chain,[10] we know

$$\Pi = \Pi\ M$$

Solving the above simultaneous equations, we obtain

$$\pi_1=\pi_2=\pi_3=0.2,\ \pi_4=\pi_5=\pi_6=\pi_7 = 0.1$$

We shall use the example in Figure 4 to illustrate the way to derive maximum queue length, average queue length, and waiting time.

It is obvious from Figure 8(d) that the maximum queue length (the maximum number of tokens) in places 1 and 2 is 2.

In terms of the probability of being in a state and the distributions of tokens in every state, the average number of tokens in a place is

$$\overline{N}_i = \Sigma\ n_{ij}\ \pi_j$$

where $\overline{N}_i$ = the average number of tokens in place i

n_{ij} = the number of tokens in node i at state j

π_j = the probability of being in state j.

In the example shown in Figures 8(d) and (e)

$$\pi_1=\pi_2=\pi_3=0.2,\ \pi_4=\pi_5=\pi_6=\pi_7 = 0.1$$

$$\overline{N}_1 = 1x0.2+2x0.2+2x0.2+1x0.1+1x0.1+1x0.1+1x0.1 = 1.4$$

$$\overline{N}_2 = 1x0.2+0x0.2+0x0.2+1x0.11+1x0.1+1x0.1+1x0.1 = 0.6$$

From the states in Figure 8(d), it is observable that the system is not at a peakload. Also, in general, the calculation of the probability of being in a state is not so straightforward. We therefore generalize from the result of a discrete-time Markov chain to get the probability.

In an ordinary Markov chain, a system not only has the Markov property but also is assumed to be in a state for a unit of time, taking no time to move from one state to another. We shall generalize from this that a system can stay in a state for a fixed unit of time, as formulated in the following theorem.

Theorem 2: Given the transition probability matrix M of a system and s_i, the unit of time a system stays at state i whenever the system enters the state, then the probability of being in a state of the system can be obtained by solving

$$\Pi = \Pi M$$

$$\Sigma s_i \pi_i = 1$$

where $\Pi = \pi_1, \pi_2, \cdots, \pi_n$, and $s_i \pi_i$ is the probability of being in state i.

Proof: It is trivial that $\Sigma s_i \pi_i = 1$. We shall prove $\Pi = \Pi M$. We can expand state i into s_i substates, namely states i1, i2, ⋯, ia with $a = s_i$. Let the input edges to state i in the original state diagram be the input edges to state i1; i1 has a probability of 100 percent to go to state i2, and i2 has a probability of 100 percent to go to state i2, and i2 has a probability of 100 percent to go to state i3, etc. State ia keeps the same output edges of state i as in the original diagram. Figures 9(a) and (b) show respectively state i in the original state diagram and the diagram after expansion. We expand other states similarly.

After the expansion, the system becomes an ordinary Markov chain, i.e., it stays in a state for a unit of time. Because of the way we expand the states, obviously

$$\pi_{i1} = \pi_{i2} = \cdots = \pi_{ia}$$

Using the solution for an ordinary Markov chain and substituting in the above equation, we still get

$$\Pi = \Pi M \qquad \text{Q.E.D.}$$

The example shown in Figure 10(a) illustrates the above theorem.

Theorem 2 states that we can convert Figure 10(a) to an equivalent state diagram, which is shown in Figure 10(b). And Figure 10(b) is an ordinary Markov chain, because the system stays in each state one unit of time whenever a system enters that state.

Since $\pi_{11} = \pi_{12}$ and $\pi_{21} = \pi_{22} = \pi_{23}$ in Figure 10(b), when these equations are plugged into the probability equations of the system shown in Figure 10(b), it is found that the following steps are a simplified way to find the probability of being in a state of Figure 10(a).

The probability of being in state 1 is $2\pi_1$. The probability of being in state 2 is $3\pi_2$.

Let $\Pi = \left[\pi_1, \pi_2 \right]$

$$M = \begin{bmatrix} 3/4 & 1/4 \\ 1 & 0 \end{bmatrix}$$

Then

$$2\pi_1 + 3\pi_2 = 1 \qquad (1)$$

$$\Pi = \Pi M \qquad (2)$$

Solving (1) and (2), we get

$$2\pi_1 = \frac{8}{11}, \qquad 3\pi_2 = \frac{3}{11}$$

Note also that theorem 2 can be extended easily to cover the cases that require time to traverse certain edges in a diagram. These cases will occur for transitions whose firing times have variances.

4. SUMMARY

We have introduced a complete Petri net-like model suitable for specification, simulation, and analysis of a system or a design. Based on this model, we have discussed system bottlenecks, resource utilization, peakload, and cost effectiveness of a system.

We have also introduced a way of converting a net or a timed net to a state diagram. A method of calculating the probability of a system's being in a state of a state diagram has been suggested. By this method, we can obtain the average length of any queue in a system. The relative activating frequency of a transition in a net depends on the topology of the net and is calculable. Thus, if one absolute activating frequency of a transition is known, then the absolute frequencies of all the other transitions are known. Hence, the arrival rate of tokens to a place that is equal to the sum of the absolute firing frequencies of the input transitions becomes available. Using Little's result with the obtained average queue length and the arrival rate, we get the average time a user (token) stays in a queue (place) for each queue in a system.

REFERENCES

1. J. D. Noe and G. J. Nutt, "Macro E-Nets for Representation of Parallel Systems," IEEETC, Vol. C-22, No. 8, August 1973.

2. A. Ellis, "Consistency and Correctness of Duplicate Data Base Systems," Proc. 6th ACM Symposium on Operating Systems Principles, November 1977, pp 67-84.

3. Y. W. Han, "An Approach of Program Documentation and Data Representation," Proc. 6th Texas Conference on Computing Systems, November 14 and 15, 1977, University of Texas, Austin.

4. Y. W. Han and W. L. Heimerdinger, "Theory of Fault Tolerance," 1977 Final Report, Contract No. N00014-75-C-0011, Prepared for Office of Naval Research, Arlington, Virginia 22217, December 1977.

5. G. Barnes, R. Brown, M. Kato, D. J. Kuck, D. Slotnick, and R. Stokes, "The ILLIAC IV Computer," IEEETC, Vol. C-17, 1968, pp 746-757.

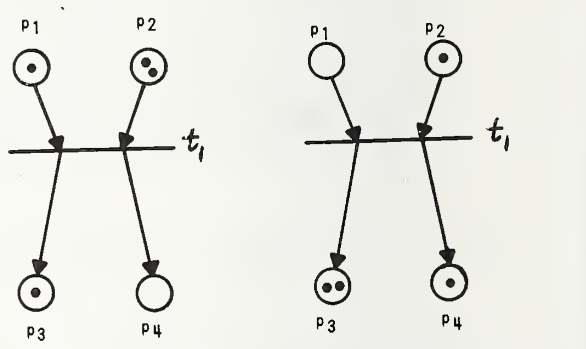
6. Y. W. Han and L. L. Kinney, "Petri Net Reduction and Verification," IEEE Repository R77-221, 1977.

7. T. Agerwala, "Towards a Theory for the Analysis and Synthesis of Systems Exhibiting. Concurrency," Ph.D. Thesis, The Johns Hopkins University, Baltimore, Maryland, 1975.

8. Y. W. Han, "Applying Graph Theory Results for System Fault Analysis," Proc. Int. Computing Symposium, December 27-29, 1977, Taipei, Republic of China.

9. C. Ramchandani, "Analysis of Synchronous Concurrent Systems by Petri Nets," MAC TR-120, Ph.D. Thesis, M.I.T., February 1974.

10. L. Kleinrock, "Queueing Systems," Vol. I: Theory, New York, John Wiley & Sons, 1975.



(a) TRANSITION 1 BEFORE FIRING (b) TRANSITION 1 AFTER FIRING

FIGURE 1 PETRI NET FIRING RULE

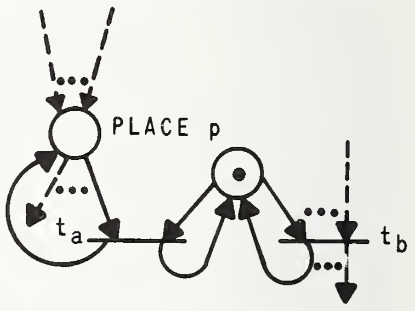


FIGURE 2 (a) NEGATION

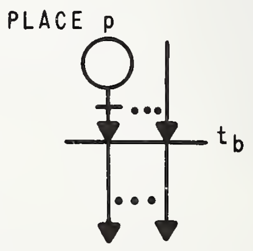


FIGURE 2 (b) NEGATION

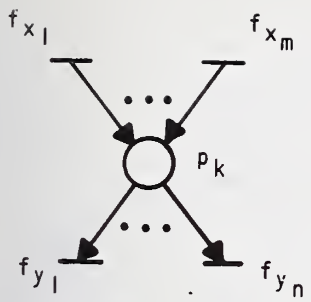


FIGURE 3 CONSERVATION OF TOKEN LAW

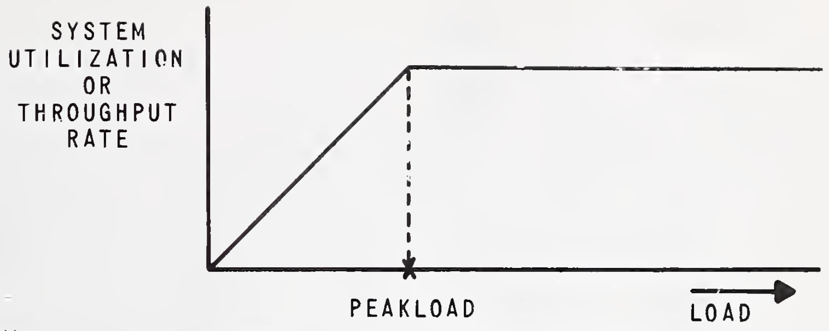


FIGURE 5 SYSTEM UTILIZATION VERSUS WORKLOAD

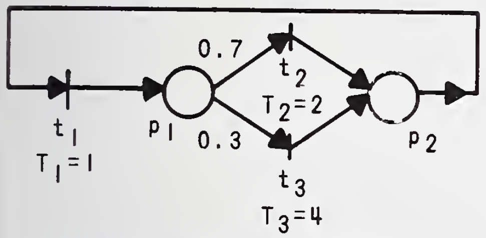


FIGURE 4 AN EXAMPLE FOR THE IDENTIFICATION OF BOTTLENECKS

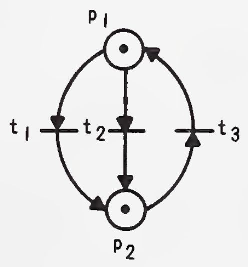


FIGURE 8 (a) A PETRI NET

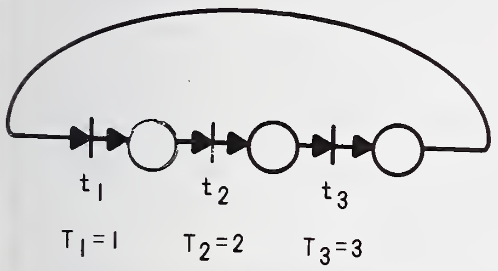


FIGURE 6 PEAKLOAD OF A SYSTEM

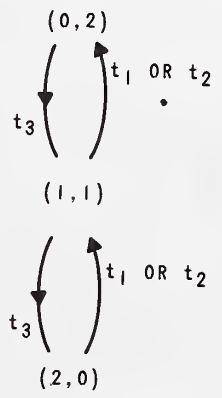


FIGURE 8 (b) STATE DIAGRAM OF 8 (a)

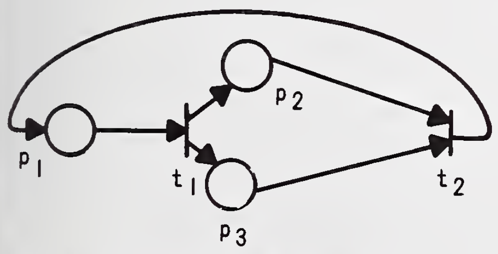


FIGURE 7 WEIGHTS OF A TOKEN IN DIFFERENT PLACES

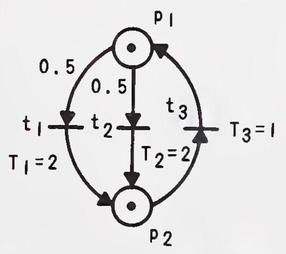


FIGURE 8 (c) A TIMED PETRI NET

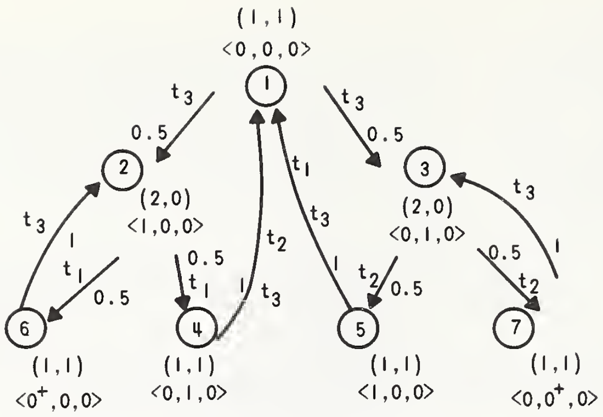


FIGURE 8 (d) STATE DIAGRAM OF FIGURE 8 (c)

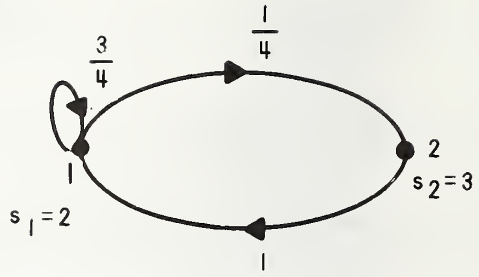


FIGURE 10 (a) A TIMED MARKOV CHAIN

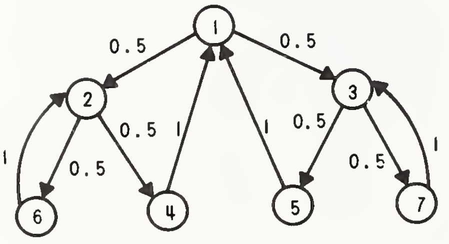


FIGURE 8 (e) SIMPLIED FIGURE 8 (d)

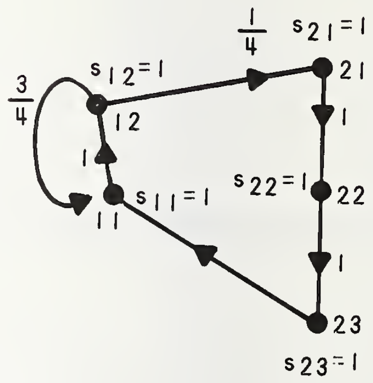


FIGURE 10 (b) THE MARKOV CHAIN AFTER EXPANSION

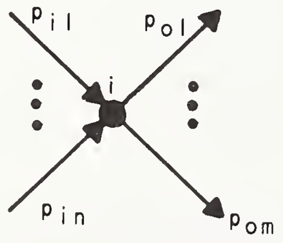


FIGURE 9 (a) ORIGINAL DIAGRAM

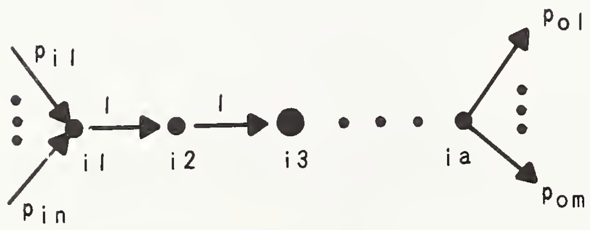


FIGURE 9 (b) DIAGRAM AFTER EXPANSION

CONTROL-THEORETIC APPROACH TO COMPUTER SYSTEMS PERFORMANCE IMPROVEMENT

Rajendra K. Jain

Digital Equipment Corporation
Systems Performance Analysis
Maynard, MA 01754

The paper presents arguments in favor of applying modern control-theoretic techniques like stochastic filtering, estimation, prediction, and time series analysis, etc., for performance optimization. It is argued that the queueing theory, which has been the most commonly used tool for computer systems modeling and performance studies, is limited in its scope due to its steady-state nature. Further improvement in performance can be obtained by dynamic optimization. Therefore, control theory provides a promising approach for performance improvement. A general methodology for formulation of operating systems resource management policies using this approach is presented. The methodology is illustrated with an example of CPU management policy.

Key words: Control theory; CPU scheduling; memory management; modeling; operating systems; performance; queueing theory; resource allocation.

1. Introduction

This paper supports control theory as a tool for dynamic optimization of computer systems performance. In spite of a few successful applications of control theoretic ideas to computer systems performance improvement, this approach has not yet become very popular. Although many modeling techniques have been proposed in the past, queueing theory is probably the most extensively used technique for evaluation and modeling of computer systems. It is a good design and static analysis tool. However, it provides little run time guidance. For dynamic (run time) guidance we need to exploit modern control-theoretic techniques such as state space models, stochastic filtering and estimation, time series analysis, etc.

The performance of a computer system is measured jointly by that of its hardware and its system software, i.e., operating system. The operating system controls the allocation and use of hardware resources such as Central Processing Unit (CPU), main memory, secondary storage, I/O devices, and files etc. Thus a prerequisite to designing a high performance system is to design high performance resource management policies.

In this paper we first argue why the control-theoretic approach seems to be a promising approach for performance improvement and what are the limitations of the classical queueing-theoretic approach. We then propose a general control-theoretic methodology to design optimum performance resource management policies. The methodology is illustrated with an example of CPU management policy.

In order to see the usefulness of control theoretic techniques, we need to first see what an operating system actually is. The two classes of analysts: the control theorists, and the queueing theorists view it differently. These two views are now described.

2. Control-theoretic View of an Operating System

For a control theorist, an operating system is a set of controllers which exercise control over the allocation of some system resource. The goal of each controller is to optimize system performance while operating within the constraints of resource availability.

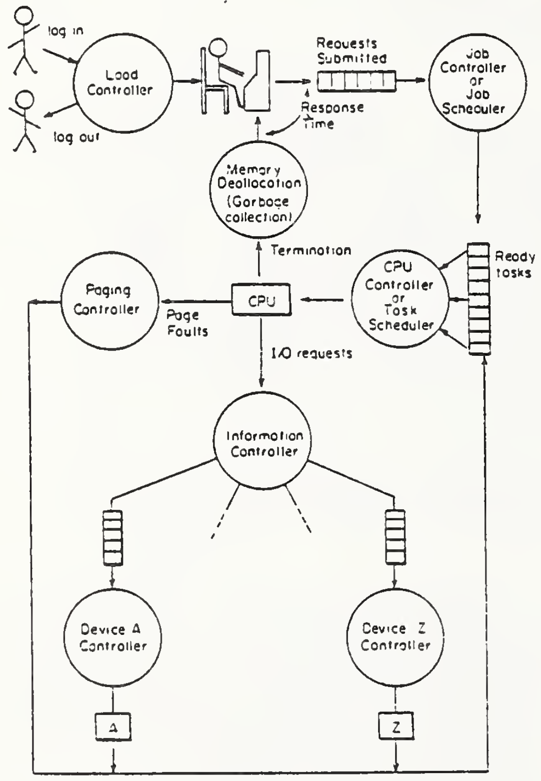


Figure 1: Control-theoretic view of an operating system

Figure 1 shows some of the components of an operating system. The Controllers are represented by circles. The "load controller" controls the number of jobs allowed to log in. The job controller (job scheduler, or high level dispatcher) controls the transfer of jobs from the "submitted" queue to the "ready" queue. This decision is based upon the availability of resources like memory, magtapes, etc. The CPU controller (task dispatcher, or low level scheduler) controls the allocation of the CPU. It selects a task from the set of ready tasks and allows it to run. The paging controller (page replacement algorithm, or memory management algorithm) controls the transfer of pages from virtual memory (disk or drum) to primary memory, and so on.

The control components of an operating system are not much different from those of other systems, except probably, in that they are non-mechanical. Therefore, there is much that can be gained from control theory in the design and modeling of these components. Unfortunately, very little control theory has been used for this purpose so far. Compared with the highly developed theory of control systems, most control algorithms used in operating systems today are "primitive".

3. Queueing-theoretic View of an Operating System

Most analytical models of computer systems used today are queueing-theoretic. From a queueing-theoretic viewpoint, each controller of the operating system is a server. Thus, an operating system is a queueing network. One very popular queueing model, called "Central Server Model", is shown in Figure 2.

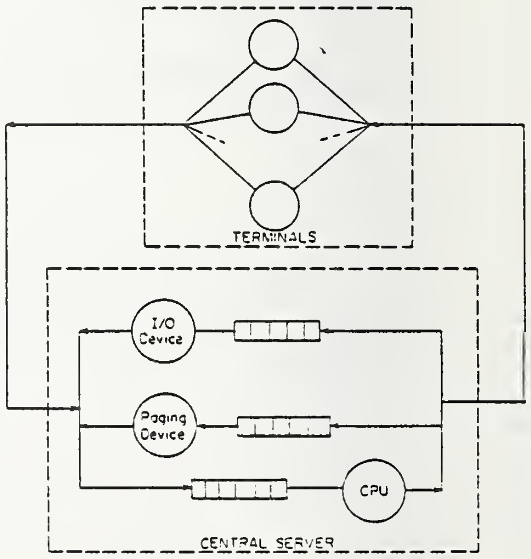


Figure 2: A queueing-theoretic view of an operating system

In this figure, circles represent servers and rectangles indicate the location of queues. Such queueing models have been used to explain many phenomena occuring in computer systems [6][1]. Typical questions that have been answered using this approach are the following :

1. What is the average throughput?
2. What is the average utilization of the CPU, I/O devices etc.
3. What is the average response time?
4. What is the bottleneck in the system (would a higher speed disk do better)?
5. What is the optimal degree of multiprogramming?

A vast amount of literature has been published to answer these and similar questions under a variety of assumptions, restrictions and generalizations. For chronological surveys and bibliographies see [15,18,11,14].

In spite of the wide applications of queueing theory, there are some inherent limitations to its usefulness.

4. Limitations of Queueing Theory

Queueing theory represents only average statistics. It tries to represent a number of jobs by the average characteristics of the class. There may be many different classes but still the "individuality" of a job is ignored. In this sense, it is a steady state analysis. It cannot satisfactorily represent time varying phenomena. Therefore, it is good only as a design time tool. It cannot be used at operation time, for which we need adaptive techniques that can adapt to the individual characteristics and time-varying behavior of jobs. To give a concrete example, a queueing model is ideal for telling whether the disk is the bottleneck in the system or whether a faster CPU will increase efficiency (both design time questions). However, once we have acquired the proper disk and CPU, it does not tell us which job from a given set of jobs should be given the CPU or the disk next. This is a dynamic decision problem, which can only be solved by the application of techniques from decision and control theory.

Queueing theory is good for modeling a computer system and, to a certain extent, its subsystems. However, when we come down to the level of a program, it cannot model its behavior (because there are no queues to be modeled). Given all the known information about a program, it cannot tell what the program behavior is likely to be in the near future. This is a prediction problem. Again, control theory must be used for this purpose.

[1]Figures in brackets illustrate the literature references at the end of this paper.

Queueing theory cannot model the interaction between the space and time demands of a program. Since the theory cannot model either the space demand behavior of a program or its time demand behavior, it certainly is inadequate for modeling the interaction between the two. Bad memory management may cause frequent page faults and may degrade the performance of an otherwise good scheduling policy. Still, the memory and the CPU allocation policies of most operating systems to date are more or less independent. This is due to a lack of clear understanding of the interaction between them. With the application of control theory we hope to remedy this situation, because, given control-theoretic models of two systems, their joint model can be obtained by modeling the cross-correlation between the two.

5. Additional Expectations from Control Theory

There are many well established control-theoretic techniques for stability, controllability, and parameter sensitivity studies that could be exploited for computer systems modeling. We hope that the control-theoretic approach will eventually lead to a better understanding of these concepts as applied to computer systems. For example, take the concept of stability. Instability in computer systems occurs in the form of excessive overhead caused by frequent switching of CPU between jobs, or by frequent oscillation of pages between main and secondary memory. The control-theoretic approach is especially suitable for stability studies, e.g., for determining the effect of sudden demand variations, or the effect of measurement delays. There are well established techniques for this purpose.

Controllability studies of computer systems could similarly help us to determine whether it is possible to reach the optimum performance state. Parameter sensitivity is already a big issue even in current queueing models. One of the major studies that investigated the applicability of queueing models to a real interactive system was conducted by Moore at the University of Michigan [17]. One conclusion of the study was that queueing models are very sensitive to parameter values which vary considerably with time and load variations. Again, control theory with its well established techniques for sensitivity analysis provides better hope.

6. Survey of Applications of Control Theory

Wilkes was probably the first to strongly advocate the exploitation of control theory for computer systems modeling. In his paper [22], he presents many arguments for applying control theory. We do not intend to duplicate those arguments here. To illustrate his ideas, Wilkes proposed a general model of paging systems.

Adaptive policies for many components of operating systems have been proposed. Dynamic tuning of allocation policies to improve throughput in multiprogramming systems has been suggested by Wulf [23]. An adaptive implementation of a load controller is described in [21]. Blevins and Ramamoorthy have investigated the feasibility of a dynamically adaptive operating system [5]. Two different techniques for adaptive control of the degree of multiprogramming have been described in [9].

The need for a control-theoretic approach was also stressed by Arnold and Gagliardi [2]. They proposed a state space formulation using resource utilization as the state variables. Using correlation properties of the memory demand behavior of programs, Arnold [1] has investigated the applicability of the Wiener filter theory to the design of a memory management policy. A dynamic programming approach to memory management and scheduling problems is described in [12,13]. A survey of some early applications of statistical techniques to computer systems analysis can be found in [3].

7. A General Control-Theoretic Methodology for Resource Management

Control systems theory and its related disciplines like stochastic filtering, estimation, prediction, and time series analysis, etc., have many possible applications to operating systems performance modeling. One application is to design resource management policies. We propose the following general control-theoretic approach to the formulation of resource management policies for operating systems.

1. In order to develop a resource management policy, model the corresponding program behavior as a stochastic process.
2. Using identification techniques and empirical data, identify a suitable model structure for the process[2] and estimate typical values of model parameters.
3. Based on the model, formulate a prediction strategy for the stochastic process, and hence a resource management policy.

The policy so obtained is dynamic in the sense that it varies the allocation of the system resource to a user job depending upon the recent past behavior of the job. It, thus, provides the run time optimization not possible with the queueing theory approach. Also, notice that the individuality of the job is fully exploited. The key step in the approach is the formulation of the stochastic process model in such a way that the allocation problem reduces to a prediction problem. We illustrate the methodology by formulating a CPU management policy. A control-theoretic memory management policy is described in [10]. Policies for allocation of other shared resources (e.g., disks) can be, similarly, formulated.

8. Example - A CPU Management Policy

The problem of CPU management is that of deciding which task from among a set of ready tasks be given the CPU next. In the literature this problem is also referred to as low level scheduling, short term scheduling, or task dispatching. There has been a considerable amount of work on designing scheduling strategies for optimizing different cost criteria, single or multiprocessor strategies, and for different precedence constraints among the jobs [7]. A common underlying assumption in all these researches is that the CPU time required by each job is known. For example, the simplest scheduling problem is that of scheduling n independent tasks with known CPU time requirements of $t_1, t_2, \ldots, t_n$ respectively on a single processor in such a way as to minimize average finish time for all users. If the jobs were scheduled in lexicographic order (i.e., 1,2,...n), the average finish time would be

---

[2]The term process is used here exclusively in the control-theoretic sense of stochastic process. To avoid confusion, the term task is used to denote computer processes e.g., we say "ready tasks" instead of "ready processes".

$$R = \frac{1}{n}\sum_{i=1}^{n}(n-i+1)t_i \qquad (1)$$

A very well known solution to this problem is due to Smith [19]. This solution is called "SPT" or Shortest Processing Time rule i.e., the jobs are given the CPU in the order of non-decreasing CPU demand. This assumes that all the tasks arrive simultaneously and are ready for processing at the same time. If tasks arrive intermittently, the optimal strategy is still basically the same. At each point in time one makes the best selection from among those jobs available, considering only the remaining processing time of the job that is currently being executed. This generalization of SPT is called the Shortest Remaining Processing Time (SRPT) rule [20].

In the case of Line printer scheduling, the service time requirements can be predicted reasonably accurately from the size of the file to be printed or by counting the number of linefeeds and formfeeds if necessary. However, in the case of the CPU, there is no known method of predicting the future CPU time requirements of the job. This makes SRPT and all similar scheduling strategies unimplementable.

In the absence of knowledge of program behavior, the operating system designer is left to use his own ad hoc prediction strategy. One such strategy is to assume that all the tasks are going to take the same (a fixed quantum of) time. The tasks are, therefore, given the CPU in a round robin fashion for the fixed quantum of time, and if a task has not completed by the end of the quantum, it is put back on the run queue. Clearly full-information strategies like SRPT perform better than no-information strategies like the fixed-quantum round robin. In order to develop a CPU management policy using the methodology described in the previous section, we proceed as follows:

Step 1: Modeling:
The first step is to model the CPU demand as a stochastic process, so that we can predict the future demands of a job from its past behavior. The kth CPU burst of ith job is modeled as a random variable $z_i(k)$. One way of representing a stochastic process is to model it as the output of a control system driven by white noise (see Figure 3).
Thus, as seen by the CPU scheduler, the program is like a control system which generates successive CPU demands. A general time series model for such a stochastic process is given by the following equation:

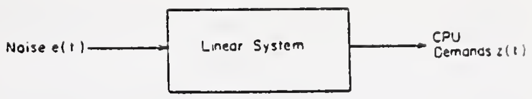


Figure 3: CPU demands modeled as a stochastic process

$$z_i(t) = f(z_i(1), z_i(2), \ldots, z_i(t-1), e_i(1), e_i(2), \ldots, e_i(t)) \qquad (2)$$

Where $z_i(t)$ represents tth CPU burst and $e_i(t)$ is the tth random shock. The subscript i is the job number; its presence indicates that the CPU demand of a job are related to previous demands of the same job and not to that of other jobs or to that of a class of jobs. A linearized and time invariant form of the above equation is the well known ARMA(p,q) model :

$$z_i(t) = w_i + a_{1i}z_i(t-1) + \ldots + a_{pi}z_i(t-p) + e_i(t) - b_{1i}e_i(t-1) - \ldots - b_{qi}e_i(t-q) \qquad (3)$$

We choose this formulation to model the CPU demand behavior of programs, because there are well established techniques to find such models from empirical data. Once a suitable ARMA model is found, it is easy to convert it to other models (e.g., state space model), if necessary.

Step 2: Analysis:
The second step is to identify a suitable model for the demand process using identification techniques and empirical data. Therefore, we conducted an experiment at Aiken Computation Laboratory, Harvard University. The experiment consisted of 19 different runs spread over a month. Each run consisted of randomly selecting a user and watching his history for a period of about 45 minutes. The data was later translated to produce the CPU demand processes of individual programs.

The identification analysis showed that the CPU demand process is a stationary process, and is best represented by a non-zero mean white noise model:

$$z_i(t) = \bar{z}_i + e_i(t) \qquad (4)$$

Where $\bar{z}_i$ is the mean of the process, and

$e_i(t)$ is white noise. See [10] for further details on the experiment and the analysis.

Step 3: Policy formulation:
The third step is to formulate a prediction strategy based on the above model and hence a resource management policy. Since, $e_i(t)$ is uncorrelated zero mean noise, it cannot be predicted, and the best estimate of the future CPU demand is its mean value, i.e.,

$$\hat{z}_i(t) = \bar{z}_i \qquad (5)$$

where $$\bar{z}_i = \frac{1}{N}\sum_{k=1}^{N} z_i(k) \qquad (6)$$

The problem in using the above formula is that $\bar{z}_i$ can be calculated only after all values of $z_i(t)$, $t=1,2,\ldots,N$ are known. What we need now is an adaptive technique to calculate $\bar{z}_i$ and update it each time a new observation is obtained. Some of the possible adaptive methods are discussed below.

1. Current Average : Average of all values observed up to t-1.

$$\begin{aligned}\bar{z}_i(t) &= \frac{1}{t-1}\sum_{k=1}^{t-1} z_i(k) \quad t>1 \qquad (7)\\ &= \frac{t-2}{t-1}\bar{z}_i(t-1) + \frac{1}{t-1}z_i(t-1)\\ &= (1-a_t)\bar{z}_i(t-1) + a_t z_i(t-1)\\ &\qquad \text{where } a_t = \frac{1}{t-1}\end{aligned}$$

Here, $\bar{z}_i(t)$ denotes the current estimate of the mean.

2. Exponentially Weighted Average :

$$\bar{z}_i(t) = (1-a)\bar{z}_i(t-1) + a z_i(t-1) \qquad (8)$$

This is a specialization of case 1 above with a_t taken to be a constant rather than a variable.

3. Average of the last n values : n=constant

$$\bar{z}_i(t) = \frac{1}{n}\sum_{k=1}^{n} z_i(t-k) \qquad (9)$$

Regardless of which formula is used for prediction, the scheduling algorithm basically remains the same : the job with the shortest predicted remaining time is selected for CPU allocation. We call it SPRPT (Shortest Predicted Remaining Processing Time) algorithm.

Notice that for prediction we do not require any extra bookkeeping other than what is already done by the operating system. Most operating systems record CPU time used by programs for accounting and billing purposes.

In fact, it turns out that Dijkstra's T.H.E. operating system [16] does use a scheduling algorithm based on scheme 2 above (exponentially weighted average). However, the algorithm was based on the simple argument that I/O bound program should be given preferential CPU allocation, and that a program should not be classified as CPU bound simply because it took large CPU time during the last burst. The exponential weighted average was thought to be a better indicator of CPU boundedness.

9. Conclusions and Directions for Future Research

Most resource management problems are basically prediction problems. Therefore, we advocate the use of modern stochastic control theory to formulate operating systems resource management policies. In this paper, we have proposed a general approach to the prediction of resource demands of a program based on its past behavior.

We exemplified the approach by applying it to the problem of CPU management. Application to memory management is described in [10]. There a new page replacement algorithm called "ARIMA" is proposed. Even though the origin of the algorithm lies in complex control-theoretic ideas, its final implementation is very simple. Moreover, it turns out that many conventional page replacement algorithms like the working set algorithm [8], Arnold's Wiener filter algorithm [1] and the independent reference model [4] are special cases of the ARIMA algorithm. The control-theoretic derivation of conditions under which these algorithms are optimal is also presented.

One interesting outcome of the research reported here is that our control-theoretic approach also provides an explanation for many previously described policies that are based on completely non-control-theoretic principles.

There are many avenues along which the research reported in this paper can be extended. The first possibility is to investigate the problem of joint management of CPU and memory. Generally, CPU and memory demands are modeled as independent processes. Strictly speaking this is not true; the CPU demand is affected by the memory policy. For example, a bad memory policy may result in frequent page faults causing tasks to be descheduled prematurely.

The control-theoretic approach can be extended to the management of other resources, e.g., disks. The disk scheduling policy can be optimized if the disk demand behavior of programs is predicted in advance. Also as mentioned previously, stability and controllability studies of computer systems require research.

The approach can also be used for the modeling of other systems. For example, in a database, the record access patterns can be modeled as a stochastic process and its prediction used to determine the optimal organization and, hence, the reorganization points of the database. In the case of computer networks, the arrival patterns of packets at a node can be modeled as a binary stochastic process. The forecast of future packet arrivals can then be used for flow control or to avoid congestion in the network.

The essence of our philosophy in this paper is that control-theorists have made good use of computers to develop better and faster modeling, estimation and prediction techniques. It is now time for computer scientists to use these techniques to enhance the cost-effectiveness of computer systems.

REFERENCES

[1] Arnold, C. R., A control theoretic approach to memory management, Proceedings Ninth Asilmar Conference on Circuits, Systems, and Computer, Pacific Grove, Calif., November 1975.

[2] Arnold, C. R., and Gagliardi, U. O., A state-space formulation of the resource allocation problem in computer operating systems, in Proc. 8th Asilmar Conf. on Circuits, Systems, and Computers, Pacific Grove, Calif., December 1974, pp. 713-722.

[3] Ashany, R., Application of control theory techniques to performance analysis of computer systems, in Proc. 6th Asilmar Conf. on Circuits, Systems, and Computers, Pacific Grove, Calif., November 1972, pp. 90-101.

[4] Aho, A. V., Denning, P. J., and Ullman, J. D., Principles of optimal page replacement, JACM, Vol. 18, No. 1, January 1971, pp. 80-93.

[5] Blevins, P. R., and Ramamoorthy, C. V., Aspects of a dynamically adaptive operating system, IEEE Trans. Comput., Vol. C-25, No. 7, July 1976, pp. 713-725.

[6] Buzen, J. P., Queueing network models of multiprogramming, Ph. D. Thesis, Harvard University, Cambridge, Mass. 1971.

[7] Coffman, E. G., Computer and Job Shop Scheduling Theory, John Wiley & Sons, 1976.

[8] Denning, P. J., The working set model for program behavior, CACM, Vol. 11, No. 5, May 1968, pp. 323-333.

[9] Denning, P. J., Kahn, K. C., Leroudier, J., Potier, D., and Suri, R., Optimal Multiprogramming, Acta Informatica, Vol. 7, fasc. 2, 1976, pp. 197-216.

[10] Jain, R. K., Control-theoretic formulation of operating systems resource management policies, Ph. D. Thesis, Harvard University, Cambridge, Mass. 1978, available as Aiken Computation Lab Report TR-10-78.

[11] Kleinrock, L., Queueing systems, Vol. 2: Computer Applications, ch. 4, New York: Wiley-Interscience, 1976.

[12] Lew, A., Optimal resource allocation and scheduling among parallel processes, in Parallel Processing, Tse-Yun Fung, Ed., Springer-Verlag, Berlin, 1974.

[13] Lew, A., Optimal control of demand-paging systems, Information Sciences, Vol. 10, No. 4, 1976, pp. 319-330.

[14] Lipsky, L., and Church, J. D., Applications of a queueing network model for a computer system, Computing Surveys, Vol. 9, No. 3,

September 1977, pp. 205-221.

[15] McKinney, J. M., A survey of analytical time-sharing models, Computing Surveys, Vol. 1, No. 2, June 1969, pp. 105-116.

[16] Mckeag, R. M., and Wilson, R., Studies in Operating Systems, Academic Press 1976, Chapter 4.

[17] Moore, C. G., III, Network models for large-scale time-sharing systems, Ph. D. Thesis, Univ. of Michigan, Ann Arbor, 1971.

[18] Muntz, R. R., Analytic modeling of interactive systems, Proc. IEEE, Vol. 63, No. 6, June 1975, pp. 946-953.

[19] Smith, W. E., Various optimizations for single-stage production, Naval Res. Logist. Quart., 3(1956) pp. 59-66.

[20] Smith, D. R., A new proof of the optimality of the shortest remaining processing time discipline, Operations Research, Vol. 26, No. 1, Jan-Feb 1978, pp. 197-199.

[21] Wilkes, M. V., Automatic load adjustment in time-sharing systems, in Proc. ACM-SIGOPS workshop on System Performance Evaluation, Harvard University, Cambridge, Mass. April 1971, pp. 308-320.

[22] Wilkes, M. V., The dynamics of paging, Computer Journal, Vol. 16, No. 1, February 1973, pp. 4-9.

[23] W. A. Wulf, Performance monitors for multiprogramming systems, in Proc. 2nd Symp. on Operating Systems Principles, Princeton Univ, October 1969, pp. 175-181.

CPEUG 78

PREDICTION PART II: QUEUING-BASED

AN INVESTIGATION OF SEVERAL MATHEMATICAL MODELS OF QUEUEING SYSTEMS

Rollins Turner
Digital Equipment Corporation
Maynard, MA 01754

A number of simple mathematical models were used to predict average response time of a timesharing system. The target system was a very simple trace driven simulation model, but the workloads were trace files obtained from a real system in normal operation. As such, the workloads were characterized by very high coefficients of variation in resource demands and think times. Mathematical models of the system included independent arrival models (M/M/1 and M/G/1, closed network models) admitting product form solutions, and a more general Markov model. Only the final model produced reasonable accuracy.

A number of experiments were performed, in an effort to determine what properties of the system being modeled were responsible for the failure of all the simple mathematical models. The large variance in CPU time and the fact that the system was a closed network were found to be critical factors, and appeared to be the major causes for failure of models that do not take them into account.

1. Preface

Both the literature and the folklore of performance evaluation are full of contradictions about the efficacy of various methods of modeling a complex timesharing system. One feels that there is a strong bias in the literature toward reporting successes and forgetting failures. In fact, experience has shown that a good paper, with a very positive tone can often be gleaned from a project that was a total failure in terms of practical value. Faced with this lack of reliable information about the practical benefits of various mathematical techniques, I decided to make an independent assessment of several of them. This paper gives the results of that project.

2. Framework for Evaluation

The purpose of this project was to investigate various approaches to predicting system response time mathematically, in terms of their accuracy for realistic workloads. The workloads to be used as the basis for this investigation were obtained from measurements on a moderately large timesharing system, taken during its normal prime time operation over several days. I felt that it would be asking too much of any available mathematical models to predict the actual system response times (which were available on the same files as the workloads), or even those of a detailed simulation. Rather, I attempted to predict the results of running these workloads through a very simple simulation model. In addition, I tried to obtain some insight into the effects of various properties of the workloads on the accuracy of the mathematical models.

3. The Workload Descriptions

The workload descriptions were derived from trace files, which were available as a result of earlier work. Each trace file has one record corresponding to each user interaction completed during the period of observation. Typically they cover 10 to 15 minutes of operation, including 2000 to 3000 interactions. For each interaction the trace file contains a number of different items of data. The only items used in this study were:

1. User think time

2. CPU time

3. Core memory used

4. Amount of disk I/O.

For most of the work reported in this paper, only CPU time and think time are relevant.

Table 1 includes a summary of statistics for each of the trace files.

4. The Simulation

A very simple timesharing system simulation was used to provide a "target" for the mathematical models. The model is a closed queueing network in which all service is first come first served. There is a fixed number of simulated users (40). Each time an interaction is completed for a user, the next record is obtained from the trace file, and the corresponding service request is simulated.

At the start of each interaction, the user remains idle for the specified think time. Then he submits a request for service, specifying a core memory amount, CPU time, and amount of disk I/O. The request enters a Core Assignment Queue until the required amount of core memory is available. Next the request enters a Swapin Queue where it waits until the swapping device can read in the specified amount of storage. Next the request enters the CPU queue. Upon receiving the specified amount of CPU time, the request will enter the Disk I/O queue, providing the amount of disk I/O is nonzero. If the disk amount is zero, the interaction is complete and a new trace record is obtained for this user. If disk I/O is required, the request is handled as soon as it reaches the head of the queue. The disk I/O is assumed to be sequential, beginning at a random point on the disk. Seek time, rotational latency, and transfer time are simulated.

The program to perform this simulation was written in SIMULA, in a very straightforward manner. For future reference, the name of the program was TIMSHR.SIM.

For the purpose of the studies reported in this paper, all simulations were run with a large amount of core, ensuring that users never had to wait to be swapped in.

Also, swapping and I/O are relatively insignificant compared to CPU time. The output of interest for each run is average response time -- defined as the mean time from submission of a request for service until completion of service. The results of the simulation runs are included in Table 1 along with the trace file statistics.

Table 1. Response times from program TIMSHR.SIM, with unlimited core and 40 users.

Trace ID	Number Interactions	Think Time Avg	Think Time Var	CPU Time Avg	CPU Time Var	Response Time - Avg
1	2919	7.44	964	.11	1.96	1.43
2	2469	7.64	1154	.16	2.36	1.98
3	1171	9.94	1406	.45	109.32	13.60
4	3434	6.99	1222	.12	4.41	1.94
5	3445	6.93	1088	.10	.77	1.14
6	2788	8.32	1415	.08	.31	.55
7	3153	8.65	1837	.11	2.13	1.17
8	3642	7.48	1354	.13	.88	1.35

5. Independent Arrival Models

The simplest approach to predicting response times is to assume independent arrivals according to a Poisson process. If we assume service times are exponentially distributed, we can use the classical queueing theory result ([5] p. 98):

$$T = \frac{1/\mu}{1 - \rho} \qquad (1)$$

where T is average response time
μ is average service rate
ρ is utilization factor λ/μ
λ is average arrival rate.

We may assume an arbitrary distribution for service time if we want to include the effect of variance in service times. In this case we would use the Pollaczek-Khinchin formula ([5] p.190):

$$T = \left(\frac{1}{\mu}\right) \left[1 + \frac{\rho(1 + C_b^2)}{2(1 - \rho)}\right] \qquad (2)$$

where C_b is the coefficient of variation for the service time.

To use these formulas, we compute the mean arrival rate by dividing the number of interactions by the total amount of time simulated in processing a trace file. μ and C_b are available directly from the trace data. The results of these calculations on the trace file data are given in Table 2. The simulation results are also given for comparison. We see that neither model provides a reasonable approximation to the simulation results.

Table 2. Comparison of predictions for M/M/1 and M/G/1 system to simulation results.

Trace ID	Arrival Rate	CPU Time Avg	CPU Time Var	MM1 Response	P-K Response	Simulation Response
1	4510	.11	1.96	.218	8.935	1.43
2	4158	.16	2.36	.478	14.977	1.98
3	1699	.45	109.32	1.911	395.605	13.60
4	4479	.12	4.41	.259	21.523	1.94
5	4957	.10	.77	.198	3.933	1.14
6	4510	.08	.31	.125	1.196	.55
7	4073	.11	2.13	.199	8.013	1.17
8	4582	.13	.88	.322	5.211	1.25

Examining the results in Table 2, we might ask <u>why</u> these simple formulas fail so dramatically to predict the simulation results. The answer is clear in the case of the M/M/1 formula. The very large variance in CPU time has an extremely detrimental effect on response time as compared to a system with the same mean service requirement but a smaller variance. But why then is the P-K formula off in the <u>other</u> direction? Evidently a workload from a finite population is "easier" in some sense than one from independent arrivals, when the parameters μ , λ , and C_b are identical. (By "easier" I mean the same server can handle it with a faster average response time.) We shall examine this hypothesis again later.

The major conclusion from this section is that the traditional "simple formulas" are not reliable for predicting the behavior of a system such as our simple timesharing model -- at least when the service distribution has a large coefficient of variation.

6. Modified Independent Arrival Models

One objection that arises when we think about the preceding independent arrival models is that they do not reflect in any way the number of users in the timesharing model. Another more subtle problem is that we used the results of the simulation run in order to compute the average arrival rate. In some sense this seems to be cheating. Really we would much prefer to use only information available as <u>input</u> to the simulation model in trying to predict its results. The "modified" independent arrival models discussed in this section are the result of an attempt to deal with these problems.

If we do not want to use the simulated run time to compute the arrival rate, we must get it from the number of users, think time, and CPU time. If we <u>knew</u> the response time we could easily compute the average arrival rate as

$$\lambda = \frac{\text{Nr Users}}{\text{think + response}} \quad . \qquad (3)$$

Indeed that is effectively what we were doing in the previous section. Implicitly, we were using the average response time from the simulation to compute the average arrival rate from which we computed our predicted response time.

So, from response time we can compute arrival rate, and from arrival rate we can compute response time. This suggests an iterative approach, in which we use the results of one calculation as input to the other, hoping the results will converge.

Programs were written to use this technique with both the M/M/1 and the M/G/1 formula. Results of running these programs with the trace file data are given in Table 3. Comparison of Table 3 with Table 2 shows that the iterative P-K predictions are consistently smaller, and better, than those done with a specified arrival rate. Unfortunately, they are still too far from the correct values to be of any use.

Interestingly the iterative M/M/1 predictions are larger than the fixed M/M/1 values, which again is the right direction. But they too are not even close to the correct values.

From the results just discussed I conclude that the "modified" independent arrival models described in this section are not useful for the intended purpose.

Table 3. Comparison of Iterative M/M/1 and Iterative M/G/1 to simulation results.

Trace ID	Avg Think	CPU Time Avg	CPU Time Var	Iterative MM1	Iterative MG1	Simulation Response
1	7.44	.11	1.96	.256	5.01	1.43
2	7.64	.16	2.36	.690	6.40	1.98
3	9.94	.45	109.32	9.007	51.14	13.60
4	6.99	.12	4.41	.347	8.44	1.94
5	6.93	.10	.77	.227	2.81	1.14
6	8.32	.08	.31	.127	1.10	.55
7	8.65	.11	2.13	.217	4.83	1.17
8	7.48	.13	.88	.384	3.32	1.25

7. Network Model

The next attempt at predicting the timesharing system simulation results used a mathematical model for a closed queueing network. Mathematical methods are available to predict the performance of a network similar in form to the simulation model [1]. An available program, ASQ [4], was used to carry out these calculations. The users' think times were modeled as a single server with state dependent service rate. The CPU and disk were modeled as FCFS servers with exponentially distributed think times. The means were set to match the corresponding means from the trace tape. Core assignment, swapping, and disk I/O were not considered in the ASQ model. Core assignment did not enter into the simulation results, as all runs were made with unlimited core available. Swapping and disk I/O were not significant in the simulation runs.

The results of the ASQ runs are given in Table 4, along with the corresponding simulation results and iterative M/M/1 results. We see that the ASQ results are very close to the Iterative M/M/1, and that both differ significantly from the simulation results. The obvious flaw in this model is the assumption of exponential service distributions. Unfortunately, the mathematical solutions do not admit other distributions with FCFS service discipline.

Table 4. Comparison of ASQ results to Iterative M/M/1 and Simulation results.

Trace ID	Avg Think	Avg CPU	ASQ Response	Iterative MM1	Simulation Response
1	7.44	.11	.24	.256	1.43
2	7.64	.16	.58	.690	1.98
3	9.94	.45	8.08	9.007	13.60
4	6.99	.12	.314	.347	1.94
5	6.93	.10	.213	.227	1.14
6	8.32	.08	.126	.127	.55
7	8.64	.11	.209	.217	1.17
8	7.48	.13	.36	.384	1.25

8. Sensitivity Studies

At this point in the project I tried to determine what properties of the actual trace file cause the simple mathematical models to fail. I had a good explanation for the M/M/1 and ASQ models: the large variance in CPU time, which violates major assumptions of the mathematical models. But what about the Iterative M/G/1? Presumably the Pollaczek- Khinchin formula is accurate if its assumptions are met. What is it about these trace files that so seriously violates the assumptions? I formed a number of hypotheses and tested each in turn on relatively small amounts of data in an attempt to answer this question.

Hypothesis 1: The synthetic trace files are too short to reflect all the bad effects predicted by the P-K formula. Perhaps the simulated system does not reach equilibrium, and the average queue length is steadily increasing.

Test: Run the simulation with progressively longer trace files and see if the results get closer to what the model predicts. Synthetic trace files with hyperexponentially distributed CPU times were generated. The first had 5000 records, the second 20,000 - resulting in more than an hour of simulated system operation.

Results: The simulated response times were still significantly faster than predicted by the Iterative P-K model. Ratios were in the same range as the earlier tests with smaller numbers of interactions.

Conclusion: Reject Hypothesis.

Hypothesis 2: The Iterative M/G/1 model fails to predict the simulation results when the pool of "thinking" users is relatively small.

Test: Run simulations with synthetic trace files having various values of average CPU time. This will result in various values of the average number of "thinking" users. See if error varies accordingly.

Results: See Table 5.

Table 5. Simulation Results

Avg CPU	C_b*	Avg Think	Avg Nr Thinking	Avg Response	Iterative MGI Response
.02	13.15	10.14	39.2	.21	.10
.04	11.26	10.14	38.5	.40	.40
.08	10.33	10.14	36.5	.99	1.63
.12	10.01	10.14	34.4	1.68	3.50
.16	9.85	10.14	32.2	2.51	5.75

*coefficient of variation for CPU tume.

Conclusion: Cannot reject the hypothesis. However the error seems to be growing much faster than the average number of thinking users is decreasing. This suggests that some other mechanism is actually the major factor.

Hypothesis 3: The Iterative M/G/1 model gets worse (at predicting simulation results) as C_b, the coefficient of variation of the service time, increases. This seems almost inevitable since M/G/1 with a C_b of 1 is equivalent to M/M/1, and we know that the Iterative M/M/1 is accurate for a synthetic load with exponentially distributed workload factors.

Test: Run simulations with synthetic trace files having various values of C_b, but all other parameters alike. Compare results to Iterative M/G/1 predictions.

Results: See Table 6.

Table 6. Effect of increasing C_b on error in Iterative M/G/1 prediction.

Avg CPU	C_b	Avg Think	Simulation Response	Iterative MG1 Response
.10	1.10	7.54	.32	.207
.11	2.26	7.54	.51	.505
.12	4.32	7.54	1.00	1.399
.11	7.71	7.54	1.48	2.714
.09	12.46	6.82	1.43	4.664

Conclusion: Accept hypothesis.

Hypothesis 4: The correlation between think time and CPU time reduces the adverse effect of high variance in both, making the M/G/1 model overly pessimistic compared to the simulation.

This is motivated by the observation that the trace file records for a specific timesharing job often consist of clusters of interactions with very short think times and short CPU times. The last interaction of the cluster has a large CPU time, and the first interaction of the cluster has a large think time. This means that we are less likely to get a sequence of interactions with short think times and large CPU times that we would be if the two distributions were independent.

Test: Remove the correlation between think time and CPU time in the trace file by "shuffling" the data among records. This was done among groups of 100 records. One hundred records were read into an array. The think time field for each record was exchanged with that of another record selected randomly from the array. CPU times were similarly shuffled. Then the 100 records were written out, and the operation was repeated on the next 100. (The actual correlation was not computed either before or after the shuffling.)

Result: There was no significant change in response time from the simulation.

Conclusion: Reject hypothesis.

Hypothesis 5: Some property of the trace file other than its mean think time, mean CPU time, and coefficient of variation of CPU time is responsible for the discrepancy between the simulation results and Iterative M/G/1 predictions.

Test: Create a synthetic trace file with exponential think times and hyperexponential CPU times. Compare simulation results to Iterative M/G/1 predictions for the values of the three parameters that actually occur in the synthetic trace.

Result: Iterative M/G/1 prediction was too large by more than a factor of two.

Conclusion: Reject hypothesis.

Now we have found that there is a really fundamental flaw in the Iterative M/G/1 model as a tool for predicting response times of a finite population system. This trace file met all the explicit assumptions of the model. It was at this point I began to think that the problem was simply that open queueing systems are different enough from closed systems that it doesn't make sense to apply the Polleczek-Khinchin formula to a closed system (at least when the coefficient of variation in service time is large).

Hypothesis 6: An open M/G/1 queueing system with a large C_b has considerably longer average response time than a closed system with the same parameters.

Test: Modify TIMSHR.SIM to generate new arrivals independently of completions. (The modified simulation program was called POISSO.SIM.) Run with same trace files and same average arrival rates as the original closed queueing system simulation.

Results: Results of the independent arrival simulations are given in Table 7. We see that the response times with independent arrivals are much larger than for the closed system. We also note that the Iterative

M/G/1 predictions are not very close to the independent arrival results. The direct use of the Pollaczek-Khinchin formula gives somewhat better estimates, but these are not close enough to be of much practical value. An observation from this test was that the average queue length with independent arrivals often exceeded the total number of users in the closed queueing system.

Conclusion: Accept hypothesis.

Table 7. Comparison of simulation results for closed system and independent arrivals. Also shown are predictions of mathematical models for parameter values of the same trace file.

Trace ID	Arrival Rate	TIMSHR Simulation[1]	POISSO Simulation[2]	Iterative MG1	P-K Formula
1	4.510	1.43	8.49	5.01	8.935
2	4.158	1.98	14.60	6.40	14.977
3	1.699	13.60	175.75	51.14	395.605
4	4.479	1.94	11.57	8.44	21.523
5	4.957	1.14	10.14	2.81	3.933
6	4.510	.55	1.45	1.10	1.196
7	4.073	1.17	9.25	4.83	8.013
8	4.582	1.25	8.15	3.32	5.213

1 Closed system simulation with 40 users

2 Open system with independent random arrivals (same trace file)

8.1 Sensitivity Studies - Summary

The purpose of the work reported in this section was to determine what properties of the trace files cause the simple mathematical models to fail. I was particularly interested in the Iterative M/G/1 model, as I had originally believed that it might be of practical value in real world situations. I found that the Iterative M/G/1 model failed not because of any exotic properties of the trace file, but simply because closed queueing systems are not the same as open queueing systems. When the coefficient of variation in service times is large, as it is in these trace files (e.g. 12), the difference is significant enough to make the Pollaczek-Khinchin formula completely useless. My conclusion is that any mathematical model which is to predict the results of the original timesharing system simulation must explicitly account for both the variance in CPU time and the finite population. The fact that the number of users is fixed means that as the queue length increases, the arrival rate decreases. This negative feedback effect significantly influences the average response time, as compared to an open system with the same overall average demands.

9. Markov Models

The final part of this project consisted of an attempt to predict the results of the timesharing system simulation by means of a Markov model [5]. The Markov model explicitly deals with the two troublesome features that plagued earlier models: the finite population and the large variance in service times.

In order to model the hyperexponential service distribution, a two stage server was used. Upon beginning service, one of the two parallel stages is chosen with probability α_1, the other with probability α_2. Each service stage has an exponential service time distribution. The first has an average service rate of μ_1 and the second μ_2. The overall service time density function is therefore the hyperexponential:

$$f(t) = \alpha_1 * \mu_1 * e^{-\mu_1 t} + \alpha_2 * \mu_2 * e^{-\mu_2 t}$$

Values for the parameters α_1, α_2, μ_1 and μ_2 can be computed from the desired parameters for the overall service time distribution. (These values are not uniquely determined.) If the desired mean service time is 1/m and coefficient of variation is C, then we have:

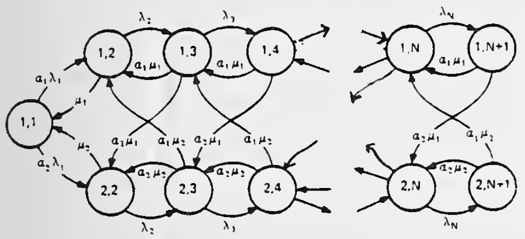
$$\alpha_1 = \frac{1}{2}\left[1 - \sqrt{\frac{C^2 - 1}{C^2 + 1}}\right] \qquad (4)$$

$$\alpha_2 = 1 - \alpha_1 \qquad (5)$$

$$\mu_1 = 2\ \alpha_1 m \qquad (6)$$

$$\mu_2 = 2\ \alpha_2 m \qquad (7)$$

We define the states of the system in terms of how many users are waiting for service, and which stage of service is in operation.



State (i,j) has j - 1 users waiting for service, one of whom is being served at service stage i. State (1,1) is an exception to this, being the idle state.

λ_i is the rate of arrival of new customers when the system is in states (1,i) and (2,i). (i - 1 users waiting.)

In these states there are N - i + 1 users thinking, with each think time drawn from an exponential (λ) distribution. (i.e. avg think time = $1/\lambda$.) Hence

$$\lambda_i = (N - i + 1)\lambda = \frac{N - i + 1}{\text{Avg think time}} \quad (8)$$

The rate of completion of service in State (i,j) is μ_i. Upon completion of service, if j > 2, the system goes to State (i,j - 1) with probability α_i (i = 1,2).

Equating the net rate of entry into each state to zero, we get 2N independent equations in 2N + 1 unknowns. To these we add the equation specifying that the state probabilities must sum to 1. The recursive solution technique of Herzog, Woo, and Chandy [3] was used to produce expressions for all state probabilities in terms of the two on the right end. Two previously unused equations yield closed form expressions for these in terms of known quantities. Given the state probabilities we can compute average arrival rate, average queue length, and average response time.

The program HERZOG.BAS was written to carry out all the calculations described above. This program unfortunately had problems with numerical accuracy. It gave reasonable answers for values of C_b close to 1, but for large values the results were obviously garbage. (e.g. Negative probabilities.) I tried a number of tricks in attempting to get around this problem, including calculating negative and positive parts of the coefficients separately. I also recoded the program in SIMULA to run on the DECsystem-10, with double precision arithmetic. However none of these measures solved the problem.

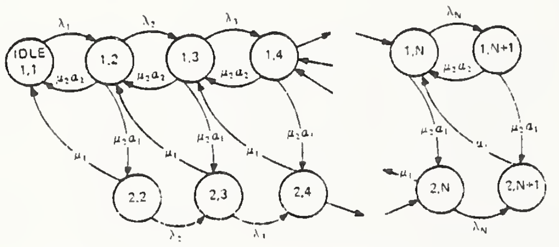
My current belief is that the problems stem from the recursive calculation of the coefficients from right to left. With realistic parameter values, the right end probabilities are very small. We go through 40 stages of calculation in computing the left end probability coefficients from those on the right end. It appears that there is no way to avoid the build up of numerical errors, using the arithmetic available in standard programming languages. As one check on the hypothesis about error build up, I ran HERZOG.BAS with very large service times, ensuring that the right end probabilities would be very large compared to those on the left end. As expected, the program gave reasonable answers for this case.

An obvious cure for the problems just described is to compute the coefficients from left to right instead of right to left. But, while the recursive substitution "falls out" very easily for right to left calculation, I was unable to find any simple procedure for calculating them from left to right. This seems to be simply an unfortunate property of the system state network topology.

The solution to this problem was a different representation of the hyperexponential service. Instead of parallel stages, a Coxian server was used. Here each customer receives service at an initial stage with exponential distribution of rate m_1. Then, with probability b he leaves the server, and with probability (1 - b) he enters a second exponential stage with mean servie rate m_2.

Theoretically values of the parameters m_1, m_2, and b can be chosen to produce any desired mean and variance in service time, but solving for the parameter values is difficult. (Cox suggests a trial and error approach [2].) I noticed, however, a close correspondence between this representation and the earlier parallel stages representations. For the system of interest (large C_b), α_1 is very small and $1/\mu_1$ is very large. Hence with low probability a customer gets a very slow server. The mean service time for the slow server is perhaps 100 times that of the fast server. In this case, it would have made very little difference to the unfortunate customer who happens to get the slow server if it had been required to also accept one service interval from the fast server. Hence we can get a good approximation to the desired overall service distribution by simply using the same parameters that we used for the parallel stages. Every customer gets service at the first stage, with mean time of $1/\mu_2$. Then, with probability α_1, he goes on to the second (very slow) stage which has a mean service time of $1/\mu_1$.

We now describe the system states as before. In state (i,j) there are j - 1 customers waiting, one of whom is in the i th service stage. As before we have state dependent arrival rates, λ_i, proportional to the number of customers not waiting for service. A state diagram for this model is shown in the figure below. This model has the fortunate property that the probability coefficients can be computed by simple recursive substitution from <u>left to right</u>.



Again following the recursive technique of Herzog, Woo, and Chandy I expressed all state probabilities as linear combinations of $P_{1,2}$ and $P_{2,2}$, and used two remaining equations to solve for them. From these values we can compute all the other state probabilities, and then all other values of interest. A program to carry out the computations was written in BASIC, and called HERZG2.BAS. Happily, that program produced reasonable answers, avoiding the numerical problems of the preceding program. The results from this program are given in Table 8. While the results are not extremely close to the target values, they are much better than those of any model considered earlier. And they are close enough to offer some hope that this model might be useful for practical purposes.

Table 8. Comparison of predictions of Markov model, program HERZG2.BAS, to simulation results.

Trace ID	CPU Time Avg	CPU Time Var	Think Time	Simulation Response	HERZG2 Response
1	.11	1.96	7.44	1.43	1.76
2	.16	2.36	7.64	1.98	2.56
3	.45	109.32	9.94	13.60	9.26
4	.12	4.41	6.99	1.94	2.19
5	.10	.77	6.93	1.14	1.31
6	.08	.31	8.32	.55	.69
7	.11	2.13	8.65	1.17	1.71
8	.13	.88	7.48	1.25	1.65

10. Overall Conclusions

The purpose of this project was to investigate the practical value of various commonly used techniques for predicting system performance. The study was rather limited, and one should naturally be cautious about drawing definite conclusions from it. Nevertheless, it adds to the body of evidence about what techniques we should or should not trust. I have more confidence in the validity of 10 to 15 minute simulation runs using real life input data, as a result of the tests made with longer runs. The effect of correlation between think time and CPU time might be a very bothersome worry, and I now have reason to believe it is not extremely important. And, in fact, I have good reason to believe that response time can be predicted fairly well on the basis of only three parameters: average think time, average CPU time, and CPU time variance. Hence one need not worry too much about any exotic properties of real world trace files (at least in very simple timesharing systems).

The importance of numerical accuracy considerations was brought home to me again. This is a lesson I have had to relearn several times, but perhaps it will be more permanent this time. The significance of a large variance in service time was another old lesson relearned in greater depth. I knew the importance of service time variance on the M/G/1 system, as expressed so vividly by the Pollaczek-Khinchin formula. But there are more subtle effects that can also be profound. As an example, my original Markov model (HERZOG.BAS) worked beautifully with a variance of 1. It was only with larger variance that numerical accuracy problems came up, and those problems were so serious as to make the program useless for its intended purpose. Another example is the Iterative M/G/1 model. With variance of 1 this model (now actually an Iterative M/M/1) is quite accurate. With a larger variance it is useless.

Perhaps the most important conclusion is that a closed queueing system, such as our very simple timesharing model, is quite different from an open system with independent arrivals. While this fact might seem obvious, it is frequently ignored in the literature. And I am inclined to believe that it is not widely understood among practitioners of performance prediction.

RFEERENCES

[1] Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios, F.G., Open, Closed and Mixed Networks of Queues with Different Classes of Customers, Journal of the ACM, Vol. 22, No. 2, April 1975, pp. 248-260.

[2] Cox, D.R. and Smith, W.L., Queues, Chapman and Hall, London, 1961.

[3] Herzog, U., Woo, L., and Chandy, L.M., Solution of Queueing Problems by a Recursive Technique, IBM Journal of Research and Development, Vol. 19, No. 3, May 1975, pp. 295-300.

[4] Keller, T.W., Towsley, D.F., Chandy, K.M., and Browne, J.C., A Tool for Network Design: The Automatic Analysis of Stochastic Models of Computer Networks, Proc. COMPCON, 1973, p151.

[5] Kleinrock, L., Queueing Systems, Vol. 1, John Wiley & Sons, New York, 1975.

ON THE BUSY PERIOD OF A QUEUEING NETWORK OF TWO SERVICE STAGES WITH EXPONENTIALLY DISTRIBUTED SERVICE TIME

Richard K. Ma

D940, IBM Office Product Division
Austin, Texas 78759

Gary J. Stroebel

D459, IBM General Systems Division
Rochester, Minnesota 55901

The recursive formula for the expected length of busy periods of a queueing network is derived. The closed queueing network consists of two queues, one being the infinite server (IS), the other a single server, both of which have exponentially distributed service time with the First Come First Served (FCFS) queueing principle. The results are compared with the infinite source queueing model with different load conditions.

Key words: queueing network; analytic modeling; computer systems; performance evaluation.

1. Introduction

Queueing theory results on the busy periods have been derived for the infinite source (Poisson arrival) queueing model M/G/1 in Kleinrock [2]. However, since Buzen [1] has shown that the closed queueing network is a more realistic representation of the real system in most engineering disciplines and the results from the infinite source model are often inaccurate, an attempt is made to obtain the busy periods of a two-server closed queing network. Fig. 1 illustrates the M/M/1 queueing model that implies an infinite source. The finite source queueing network is depicted in Fig. 2 in which the first service stage is an infinite server queue while the second service stage is a single server queue with FCFS queueing principle.

2. Problem and Solution

In Fig. 2 the number of customers circulating the network equals the fixed population of the closed system . Since stage 1 is an infinite server, there won't be any queueing in front of this service station. The service times of all the servers are exponentially distributed. The queueing in the network is less severe than for an open system with arrival rate $n\lambda$ because of the finite population. Thus the results of the open model M/M/1 are inaccurate for calculating the busy period of service stage 2 in the closed network.

Starting from n=1, where n is the population of the closed network, a general scheme is used to obtain the busy period at service stage 2. $E(B_n)$ is used to denote the expected length of the busy period of service stage 2 with population n in the closed system. Obviously, $E(B_1) = 1/\mu$ where μ is the service rate at stage 2.

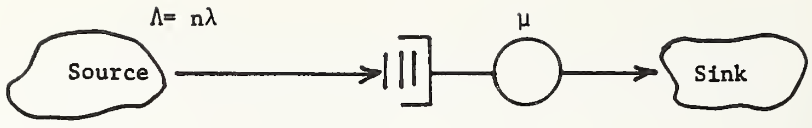


Figure 1. M/M/1 Queueing Model

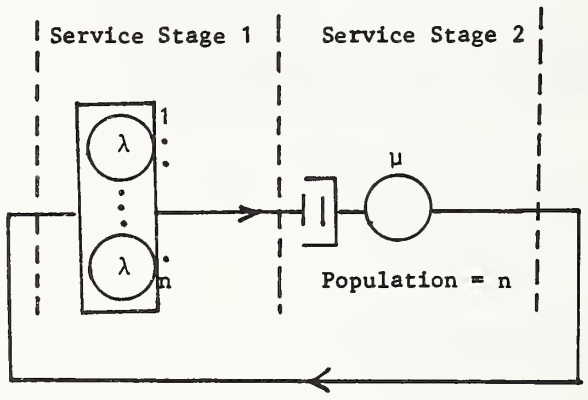


Figure 2. Finite Source Queueing Network

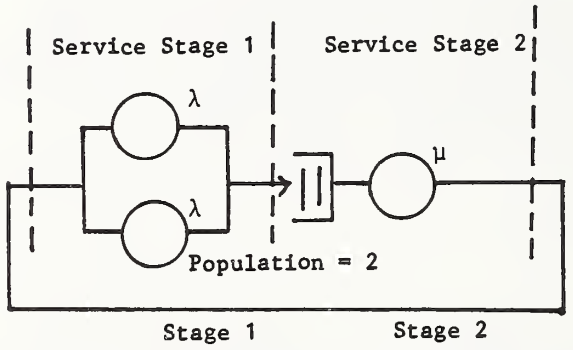
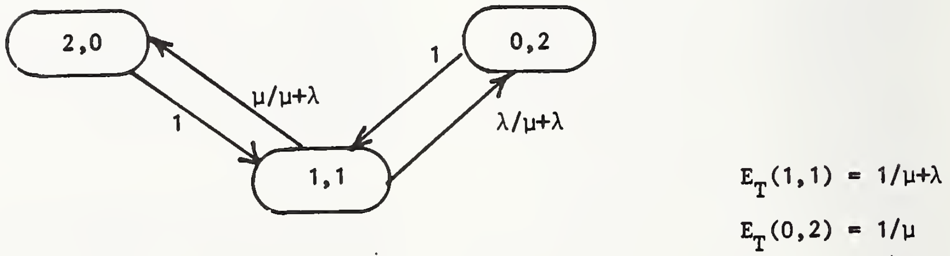


Figure 3. Closed Loop Queueing Network with Two Stages of Service and Population 2.



$E_T(1,1) = 1/\mu+\lambda$

$E_T(0,2) = 1/\mu$

Figure 4. Transistion Diagram for Figure 3.

The case of n=2 shown in Figure 3 is now studied.

Let $\underline{X} = (X_1, X_2)$ be the state of the queueing network where X_I is the number of customers at stage I. To assign n indistinguishable customers to N stations, the number of possible combinations is $\binom{N+n-1}{N-1}$. So, three possible states exist for this system in Fig. 3: (2,0), (1,1), and (0,2). The state transition diagram is as shown in Figure 4.

The numbers at the sides of the arrows refer to the probabilities of the corresponding transitions. The busy period starts at the entry to state (1,1) from state (2,0). The expected duration of state (1,1) is $1/\mu+\lambda$. When the state changes, with probabilities $\lambda/\mu+\lambda$ and $\mu/\mu+\lambda$, the system will move to (0,2) and (2,0) respectively. Also, the system will stay at (0,2) for an expected length of time $1/\mu$ and then move back with probability 1 to (1,1) again. With the memoryless property of this continuous time Markov chain, it is true that

$$\begin{aligned} E(B_2) &= (\mu/\mu+\lambda)\ (1/\mu+\lambda) + \\ &\quad (\lambda/\mu+\lambda)\ [1/\mu+\lambda + 1/\mu + E(B_2)] \\ &= 1/\mu+\lambda + (\lambda/\mu+\lambda)\ [1/\mu + E(B_2)] \\ &= 1/\mu + (\lambda/\mu)\ E(B_1) \end{aligned}$$

With n = 3, the state transition diagram is shown as in Figure 5.

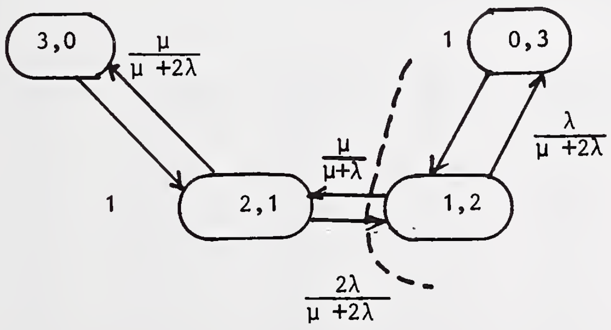
The busy period will start at the entry to state (2,1) from state (3,0). Comparing the state transition probabilities and the expected length of stay of the states to the right of the dotted line, it is found that they are exactly the same as those of the states to the right and including (1,1) in Fig. 4. The reason is that the second service stage has only one server with service rate μ.

Thus, the expected length of stay to the right of the dotted line in Fig. 5 should be just $E(B_2)$. With the same argument as used in deriving $E(B_2)$, the following is true:

$$\begin{aligned} E(B_3) &= (\mu/\mu+2\lambda)\ (1/\mu+2\lambda) + \\ &\quad (2\lambda/\mu+2\lambda)\ [1/\mu+2\lambda + E(B_2) + E(B_3)] \\ &= (1/\mu+2\lambda) + (2\lambda/\mu+2\lambda)\ [E(B_2) + E(B_3)] \\ &= 1/\mu + (2\lambda/\mu)\ E(B_2) \end{aligned}$$

Following exactly the same pattern, it can be shown that

$$\begin{aligned} E(B_4) &= 1/\mu + (3\lambda/\mu)\ E(B_3) \\ &\ \vdots \\ E(B_{n+1}) &= 1/\mu + (n\lambda/\mu)\ E(B_n) \qquad (1) \end{aligned}$$



$E_T(2,1) = 1/\mu+2\lambda$

$E_T(1,2) = 1/\mu+\lambda$

$E_T(0,3) = 1/\mu$

Figure 5. Transistion Diagram for a Two Stage Closed Queueing Network With Population = 3.

As mentioned above, the key point in deriving this recursive relation is that all the state transition probabilities and the expected lengths of stays of all states to the right of and including the starting state (eg, (n-1,1)) in the case of n customers are exactly the same as those of the states to the right of the "next to starting" state (eg, (n-1,2)) in the case of (n+1) customers. Accordingly, $E(B_n)$ can be used to derive $E(B_{n+1})$.

As $n \to \infty$, $\lambda \to 0$, $n\lambda \to \Lambda$ (1) converges

to the formula of the M/M/1 model, ie

$$E(B) = 1/\mu + (\Lambda/\mu)\, E(B)$$

$$= 1/\mu \;.\; 1/(1- \Lambda/\mu)$$

3. Computational Results and Comparisons

An APL program has been written to compute the results of the M/M/1 model and the closed queueing network model. The error percentage of using the M/M/1 model under different load conditions is listed in Table 1. In all cases the expected busy period for the M/M/1 model is longer than that of the closed queueing network.

Table 1 suggests that for the analysis of expected busy durations, the M/M/1 is an appropriate representation of the closed queueing model only when: (i) n is larger and (ii) ρ is small.

The authors thank Dr. W. M. Chow of IBM Research for his suggestions and Mr. D. Ness of IBM GSD for his review.

References

[1] Buzen, J. P. and Goldberg, P. S., "Guidelines for the use of infinite source queueing models in the analysis of computer system performance", National Computer Conference Proceedings, 1974, P 371 - P 374.

[2] Kleinrock, L., Queueing System, Vol. 1, Wiley Interscience, New York, 1975.

Table 1 Error Percentages of M/M/1 Model Approximation for the Closed Network Described Above.

n \ ρ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1	11.11	25.	42.86	66.67	100	150	233	400	900
2	5.82	13.64	24.22	38.89	60	92.31	146	257	589
4	2.99	7.23	13.26	22.01	35.09	55.74	91.47	164	389
8	1.52	3.74	7.05	12.05	19.88	32.74	55.81	104	256
15	0.82	2.03	3.89	6.80	11.53	19.63	34.78	68.01	174
30	0.41	1.03	1.99	3.54	6.13	10.79	19.98	41.24	113
50	0.25	0.62	1.21	2.16	3.79	6.80	12.96	27.91	80.74
100	0.12	0.31	0.61	1.09	1.94	3.55	6.99	15.90	50.11
150	0.08	0.21	0.41	0.73	1.31	2.41	4.81	11.24	37.33
200	0.06	0.16	0.30	0.55	0.99	1.82	3.67	8.72	30.06

PREDICTION PART III: APPLICATIONS

A MARKOVIAN MODEL OF A JOB

A. K. Agrawala
J. M. Mohr

Department of Computer Science
University of Maryland
College Park, MD 20742

A Markov model may be used to characterize a sequence of states. In this paper we explore the use of such models for modelling the sequence of job steps of a job. The job population is first divided into several clusters and then a separate Markov model is created for jobs from each cluster. The results of an experimental study conducted on a large UNIVAC machine at the Computer Science Center of the University of Maryland are presented.

Key words: workload modelling, Markov model

1. Introduction

Most of today's general purpose computers are used in a multiuser, multiapplication environment handling a workload which is a composite of all of the processing requests made by its user community. As a consequence the problem of characterizing or modelling the workload of a computer system so that all aspects of interest are captured in the model, becomes very difficult.

The first step in creating any model is to decide the degree of aggregation, i.e. to identify the smallest level of detail to be modelled. In the workload modelling context the smallest unit of work modelled has been called the workstep [1]. We may choose to model the workload in terms of instructions, subroutines, job steps, jobs, or sequences of jobs, treating any of these as the workstep. Clearly one's perspective of the problem changes as the level used for the workstep is changed. If a workstep is an instruction we get an instruction mix description of the workload. By choosing a job step as a workstep we may get a model in terms of the various system packages, such as compilers, available on the system. In order to capture the processing characteristics of the user a job has been used as a workstep in several studies [2,3,4,5].

Having decided on the size of a workstep, e.g. a job, a workload model may be formulated to account for the characteristics of the population of worksteps handled by a system. In formulating such a model the characteristics of a workstep are measured in terms of a set of features. For example, we may choose to use the resources required by a workstep as the features and describe a workstep by a vector, each element of which corresponds to the amount of a particular resource that is used. We may include as features the use of all major system resources and quantify the requirements of a workstep in terms of the CPU time, memory, and I/O to various devices etc.

A way to formulate the model of the workload is to treat the characteristics of a workstep as an n-dimensional random vector x representing the workstep's n different resource requirements. Now the model of the workload consists of a probability distribution for x defined in the n-dimensional space spanned by this vector.

Due to the large variety in the processing demands placed on the system by the users the probability distributions p(x) are usually very complex. A way to handle this

This research was supported in part by the National Aeronautics and Space Administration, Goddard Space Flight Center, under Grant NASA #NAS 5-24407 and in part by the Environmental Protection Agency under grant R805478-01-0, to the Department of Computer Science, University of Maryland, College Park, Maryland.

We wish to acknowledge the support and assistance of the University of Maryland Computer Science Center and its personnel.

complexity has been to express p(x) as a mixture distribution where

$$p(x) = \sum_{i=1}^{k} p(x/c_i)\, P(c_i) \qquad (1)$$

Here the population is grouped in k groups with the ith group having the distribution $p(x/c_i)$. Clustering has been used successfully to create models based on equation (1) in a nonparametric environment.

A model of the workload expressed as p(x), identifies a workstep in terms of a vector x, the total resource requirements of a job. When using the requirements placed on resources such as CPU, memory, and I/O devices we do not consider the time over which these demands are placed by a workstep or how they are placed. When using a job as a workstep, the vector x represents the total resource requirements of the job obtained by summing up the requirements of each individual job step. A workload model constructed using a job as a workstep does not, by design, have any details at the job step level, and hence cannot be used to infer any characteristics of the sequence of job steps in a job.

In applications such as the design of representative benchmarks, the workload must be specified at the job level and at the job step level. Only then can we construct benchmarks consisting of several jobs where each job is specified in terms of sequences of job steps.

In this paper we present a Markovian model of a job which may be used to model the job steps. Some earlier studies [6] indicate that a common Markovian model for the entire population of the jobs may not be valid. Here we model the job steps of the jobs in each cluster, or group in terms of a separate Markovian model.

Section II of this paper describes how a set of jobs may be modelled as a Markov process. In Section III the results of an experiment are described in which a large data set was clustered and each cluster was modelled as a semi-Markov process. Finally, results are presented showing the differences in the Markov processes used to model each cluster.

2. The Model

In order to perform his processing a user activates a sequence of processing requests within a job. These individual processing requests are handled separately by the system as job steps and may consist of functions such as compiling, linkage editing, or the activation of a statistical package etc. If all jobs executed the same sequence of job steps, the problem of modelling jobs at the job step level would have been very easy. In practice, however, jobs having similar resource usage profiles may have different numbers and types of job steps. In order to create a generative model of a job in terms of a sequence of job steps we need to take the sequential nature of the job steps into account.

A way to model the sequence of job steps is to treat this sequence as a Markovian sequence [7]. In order to create a Markovian model of such a sequence we need to specify a set of states. The natural set of states for job steps are the activities identified as job steps in a system. Thus we may use calls to various programs as the states of such a Markov sequence. For the discussion here let us assume that the set of states has K elements in it.

The basic assumption required for a sequence to be Markov of order one is that the transition to the next state depend only on the current state and not on the past states. In other words, a Markov sequence has a memoryless property in which knowing the current state, the past history may be ignored. With this assumption, a Markov sequence may be specified by a (k x k) transition probability matrix, P, which contains the probability of a transition from state i to state j as its (i,j)th element, P_{ij} . Clearly, since each transition probability P_{ij} is a probability, it must satisfy the requirement

$$0 \leq P_{ij} \leq 1 \qquad 1 \leq i,j \leq k.$$

Further, since the process must occupy one of K states after each transition

$$\sum_{j=1}^{k} P_{ij} = 1 \qquad i = 1,2,\ldots k.$$

Thus, we observe that all of the elements of the transition probability matrix P have values in the range (0,1) and each of its rows sum to unity.

Once we model the sequence of job steps in a job as a Markov process we may easily derive a variety of statistics about the process. As the transition probability matrix is assumed to have constant values a multi-step transition probability matrix may easily be obtained by multiplying the matrix P to itself the required number of times.

Before considering the state probabilities, we need to decide whether we are modelling the sequence as a transient Markov sequence or a recurrent Markov sequence. We recognize that each job only has a finite sequence of job steps and starts with the START state and ends in the END state. The END state may be considered as a trapping state for this sequence. To observe another sequence we have to start from the START state again. If we treat the job steps in this way we obtain a purely transient Markov model and many of the process' interesting properties, such as its steady state behavior, cannot be studied. We observe that by treating the END state, such that from there, with probability one a transition occurs to the START state, and recognize that a visit to the END state represents the termination of a job which is followed by the start of another, we can model the sequence as the sequence obtained by executing a number of jobs sequentially. Now the model we obtain has only recurrent states, (assuming all of its jobs end). For such a Markov chain we can determine the limiting state probabilities, which may be represented as a K vector π where π_i represents the probability of finding the process in state i after a transition assuming the process has been running for long enough that a starting value has no effect on its current state.

Another parameter of interest is the time (or number of steps) it takes on the average to go from the START state to the END state. Noting that END state is followed by the START state we may try to determine the first recurrence time for the START state. The average time N (i.e. the number of steps) is easily obtained as the inverse of the limiting state probability of the START state [7]. The limiting state probability vector is the eigen vector of P corresponding to its dominant eigen value and may be obtained by solving the eigen value problem.

In order to estimate the values of the elements of P and π we can use the observed data obtained from a source such as an accounting log where the job step information is recorded. The first step is to decide on the states to be used in the model. Then we may observe a sequence of job steps, recognize the state corresponding to each job step and consider the sequence of states. The maximum likelihood estimate of P_{ij} is obtained by counting the total number of transitions from state i to state j and dividing that by the total number of transitions in the sequence observed [9]. We may obtain an estimate of π by solving the eigen value problem or by counting the number of times the process was in each state. We confirmed that the values of π obtained by both of these techniques are identical when the sequence consists of a set of complete jobs.

For a single Markov model to apply to a set of jobs, those jobs should have similar characteristics. It has been reported that the jobs run on a computer operating in a multiuser, multiapplication environment are not homogeneous and in fact consist of a number of identifiable groups or clusters. As the jobs belonging to a cluster look similar to the system, they are better candidates for modelling as a Markov sequence. Therefore, to create the model of the workload of a system we have to first cluster the jobs and then create a Markov model for the jobs in each cluster.

In addition to information as to the sequence of states we may also measure the amount of time spent in each state. The holding time of a state is the amount of time spent executing the program or programs represented by the state. When these state holding times are included in the model the model becomes a semi-Markov model [8]. When a Markov model is used it is assumed that the state transitions occur at discrete instants of time and the process is examined just after a transition. In a semi-Markov model the state transition probability matrix P is augmented by a set of holding time distributions, one such distribution for each state in the Markov model.

3. An Experimental Study

In order to experimentally examine the use of a Markov model of the jobs in a real system's workload, we studied the workload on the Univac 1108 at the Computer Science Center, University of Maryland. This is one of two large UNIVAC 1100 series machines that are used by the students and the faculty of the university for educational and research computing activities. The system operates under level 33 of the 1100 Series Operating System. For the study we used approximately 2 ½ weeks worth of accounting log information. A total of 22,058 jobs were executed by the system during this period. One of the major goals of this study was to determine how much the Markov processes for the clusters differed.

3.1 Clustering

The first step was to scale and cluster the data from the log tapes. A total of eight resource related features were chosen for the clustering. These were:

1) CPU time
2) Executive service time
3) Number of programs executed
4) Amount of core used
5) SUP's *
6) Drum I/O
7) Disk I/O
8) Tape I/O

The scaling and clustering techniques used are described in [1]. We obtained a total of eleven clusters. The location of the centroid of each cluster and the number of points in each cluster is given in Figure 1. These values are presented in terms of the scaled variables where the 98th percentile of the original variable has been scaled to a value 10.0. From a comparison with results presented in [5] we note that these clusters are very similar in their characteristics to those found in [5].

3.2 The States of a Markov Model

In creating a Markov model for jobs in each cluster one must first decide on the set of states. As noted in Section 2 the programs or job steps identified by the system are well suited for this purpose. The UNIVAC system at the University of Maryland recognizes approximately 230 different system's programs, most of which are executed very infrequently. In addition, a large number of user programs may also be identified. In this study all user programs were treated as a single state and all the infrequently used programs were grouped into one of five common states:

a. Compilers
b. General file utilities
c. Tape utilities
d. Data analysis packages
e. Other utilities.

As a consequence, we used a total of 25 states for this model.

* SUP's is the measure UNIVAC uses to quantify a program's CPU time, executive service times, and I/O times. In this paper all SUP times are expressed in seconds.

3.3 The Markov Model

A separate Markov model was postulated for the jobs belonging to each of the clusters. Based on the information available from the log tapes used in this study we computed the estimates of the transition probability matrix, $\hat{P}$ as well as the limiting state probability vector $\hat{\Pi}$ for each cluster. With 25 states $\hat{P}$ is a 25 x 25 matrix. As an example, the transition probability matrix for the programs executed by the jobs contained in cluster 1 is presented in Figure 2. It shows that for a job from cluster 1 a FORTRAN cimpile is followed by a call to MAP for linkage editing with a probability of 0.518. We obtained a transition probability matrix for each of the other 10 clusters.

Another interesting characteristic of a Markov model is the limiting state probability vector. For the model formulated here it is a vector of length 25. The estimates of these vectors for each of the clusters are shown in Figure 3. From the results presented there we observe that if we were to observe a randomly selected program of a job from cluster 5, with probability 0.13 it would be a call to ED the editor. Also note that while the assembler (ASM) is not frequently used, no job in clusters 5, 6, 9, or 10 used it at all. The SPSS is used most frequently by jobs from cluster 8.

A number of interesting observations can be made from the information presented in Figure 3 and also, from that available from the transition probability matrices. However, to consider quantitatively the differences between these Markov models we need to formulate some meaningful measures of their differences. Let us consider the differences based on the limiting state probability vectors.

Two similarity measures can be used to describe the differences between a pair of k dimensional vectors, the Euclidean distance between their end points or the direction cosine between the two vectors. The direction cosine varies between 0.0 and 1.0 and describes the differences in the direction of the vectors as the cosine of the angle between them. The Euclidean distance between the endpoints of the two vectors also captures the difference due to the relative lengths of the vectors. The numeric values of these measures of difference are displayed in Figure 4. All pairwise difference measures are shown in that figure. The Euclidean distance is displayed first and

Cluster No.	# of jobs	# of Progs	SUPS	Core Blocks	CPU Time	ER & CC Charges	Drum I/O	Disk I/O	Tape I/O
1	7694	0.5	4.1	3.7	1.9	4.1	3.7	3.8	0.0
2	334	7.2	8.8	2.8	6.8	9.2	7.9	8.8	7.3
3	659	0.8	5.6	2.6	2.9	5.2	4.7	3.8	5.4
4	189	6.7	10.4	4.7	9.6	10.0	9.6	9.7	10.2
5	2797	1.8	6.0	2.4	3.2	6.5	6.1	5.4	0.0
6	1153	0.5	7.3	6.3	7.3	5.3	4.9	5.5	0.5
7	2828	5.3	8.0	3.2	6.2	8.3	8.0	7.6	0.0
8	179	1.1	7.6	6.1	7.1	6.4	7.5	5.3	7.5
9	304	3.2	7.4	2.2	4.0	7.8	6.6	6.8	6.5
10	310	0.6	9.2	4.3	7.7	7.4	5.4	4.9	11.0
11	5611	0.2	2.2	1.2	0.1	2.5	1.5	0.9	0.0

Figure 1 Scaled Means for the Clusters

	START	ASG	FREE	USER	ACCOUN	ASM	CATALO	CTS	ED	ELT	FORTRN	FURPUR	LOAD	MAP	SAVE	SPSS	RESUME	SUSPEN	STATUS	TIME	GROUP1	GROUP2	GROUP3	GROUP4	GROUP
START	.000	.209	.000	.000	.010	.001	.000	.522	.010	.090	.087	.013	.003	.001	.000	.000	.000	.000	.000	.000	.028	.000	.000	.026	.001
ASG	.000	.363	.043	.238	.002	.002	.005	.000	.064	.009	.010	.050	.004	.008	.001	.084	.002	.016	.005	.000	.004	.012	.036	.026	.014
FREE	.350	.006	.621	.003	.001	.000	.002	.000	.003	.000	.000	.002	.001	.000	.000	.000	.002	.000	.008	.001	.000	.001	.000	.001	.600
USER	.134	.035	.458	.104	.091	.000	.001	.000	.032	.002	.014	.018	.002	.027	.006	.000	.026	.001	.006	.002	.002	.010	.004	.009	.014
ACCOUN	.127	.060	.480	.016	.052	.000	.048	.000	.032	.008	.036	.008	.036	.012	.000	.000	.004	.000	.036	.000	.024	.016	.000	.008	.000
ASM	.000	.000	.000	.000	.000	.000	.000	.000	.000	.000	.077	.077	.000	.769	.000	.000	.077	.000	.000	.000	.000	.000	.000	.000	.000
CATALO	.000	.207	.153	.009	.099	.000	.063	.000	.090	.009	.000	.180	.018	.000	.000	.000	.027	.009	.054	.000	.000	.045	.000	.027	.009
CTS	.000	.356	.000	.000	.047	.000	.041	.000	.213	.001	.006	.046	.112	.000	.001	.000	.000	.017	.030	.001	.060	.035	.000	.030	.004
ED	.001	.090	.195	.012	.014	.000	.010	.000	.275	.005	.079	.094	.012	.008	.091	.000	.006	.036	.015	.009	.012	.027	.000	.007	.003
ELT	.007	.012	.044	.005	.010	.002	.000	.000	.010	.516	.308	.039	.000	.000	.015	.000	.012	.000	.000	.000	.002	.012	.000	.007	.000
FORTRN	.001	.010	.023	.000	.006	.003	.000	.000	.033	.000	.346	.019	.003	.518	.006	.000	.006	.003	.000	.013	.003	.004	.000	.001	.003
FURPUR	.011	.325	.156	.056	.007	.000	.027	.000	.144	.011	.020	.045	.007	.045	.002	.002	.033	.009	.022	.000	.007	.044	.005	.015	.005
LOAD	.000	.060	.015	.065	.005	.005	.005	.000	.322	.000	.040	.050	.111	.010	.025	.000	.000	.040	.005	.000	.106	.121	.000	.005	.010
MAP	.002	.006	.030	.623	.000	.000	.000	.000	.012	.012	.002	.274	.000	.010	.004	.000	.004	.002	.002	.000	.010	.000	.006	.000	.000
SAVE	.024	.000	.161	.016	.056	.000	.000	.000	.081	.016	.210	.056	.089	.008	.065	.000	.000	.073	.016	.008	.016	.089	.000	.008	.008
SPSS	.000	.000	.818	.000	.065	.000	.006	.000	.013	.006	.000	.000	.000	.000	.000	.045	.026	.000	.000	.000	.000	.000	.000	.019	.000
RESUME	.000	.009	.636	.037	.047	.000	.019	.000	.075	.009	.000	.056	.000	.000	.009	.000	.019	.037	.009	.009	.000	.009	.000	.009	.009
SUSPEN	.000	.277	.034	.050	.008	.008	.025	.000	.034	.042	.092	.084	.000	.017	.008	.000	.126	.025	.000	.000	.034	.000	.008	.118	.008
STATUS	.000	.119	.325	.013	.033	.000	.020	.000	.119	.000	.000	.073	.007	.000	.000	.000	.007	.026	.139	.007	.000	.033	.000	.079	.000
TIME	.000	.061	.273	.000	.030	.000	.000	.000	.061	.000	.061	.000	.030	.242	.000	.000	.000	.000	.091	.121	.030	.000	.000	.000	.000
GROUP1	.034	.028	.419	.006	.067	.017	.006	.000	.022	.000	.022	.061	.011	.151	.006	.000	.017	.006	.006	.011	.095	.000	.000	.006	.011
GROUP2	.000	.046	.137	.032	.027	.000	.027	.000	.155	.027	.027	.096	.105	.000	.027	.000	.000	.014	.027	.009	.005	.224	.000	.005	.009
GROUP3	.035	.059	.635	.035	.035	.000	.000	.000	.024	.000	.000	.012	.000	.000	.000	.000	.000	.000	.024	.000	.000	.000	.129	.000	.012
GROUP4	.024	.171	.299	.071	.028	.000	.009	.000	.033	.043	.038	.038	.019	.000	.000	.000	.033	.009	.009	.000	.009	.009	.000	.156	.000
GROUP5	.071	.000	.429	.000	.029	.000	.000	.000	.029	.000	.057	.043	.014	.057	.014	.000	.043	.014	.014	.000	.014	.029	.000	.014	.129

Figure 2 Transition Probability Matrix for Cluster 1

Cluster No.	START	ASG	FREE	USER	ACCOUN	ASM	CATALO	CTS	ED	ELT	FORTRN	FURPUR	LOAD	MAP	SAVE	SPSS	RESUME	SUSPEN	STATUS	TIME	GROUP1	GROUP2	GROUP3	GROUP4	GROUP
1	.111	.124	.279	.067	.018	.001	.008	.058	.062	.030	.050	.039	.014	.036	.009	.011	.008	.009	.011	.002	.013	.016	.006	.015	.005
2	.015	.096	.122	.099	.010	.003	.012	.012	.130	.106	.031	.149	.006	.027	.001	.000	.035	.032	.030	.008	.012	.038	.006	.012	.008
3	.099	.091	.205	.069	.008	.001	.010	.039	.050	.019	.025	.222	.004	.020	.001	.005	.009	.009	.018	.003	.003	.015	.013	.062	.000
4	.017	.088	.125	.158	.003	.005	.010	.008	.104	.046	.021	.215	.001	.031	.007	.000	.023	.024	.027	.008	.013	.028	.005	.013	.018
5	.045	.089	.213	.073	.014	.000	.009	.041	.130	.036	.066	.051	.022	.035	.023	.002	.028	.028	.018	.005	.019	.028	.003	.015	.005
6	.111	.166	.261	.123	.019	.000	.001	.030	.046	.022	.050	.050	.004	.033	.002	.017	.007	.007	.001	.005	.006	.004	.011	.014	.010
7	.020	.066	.128	.105	.010	.006	.005	.018	.146	.061	.064	.085	.015	.043	.022	.002	.040	.039	.021	.008	.032	.028	.004	.023	.009
8	.067	.087	.135	.192	.012	.002	.002	.014	.043	.020	.092	.184	.006	.035	.005	.032	.018	.018	.009	.002	.002	.020	.000	.000	.003
9	.030	.093	.164	.063	.009	.000	.009	.025	.099	.082	.024	.188	.012	.021	.002	.004	.031	.028	.018	.009	.006	.037	.005	.034	.008
10	.139	.137	.195	.207	.000	.000	.000	.006	.002	.026	.034	.163	.000	.048	.000	.000	.000	.000	.000	.004	.008	.012	.002	.018	.000
11	.193	.100	.366	.030	.030	.001	.013	.127	.029	.006	.005	.021	.007	.003	.003	.002	.001	.001	.017	.001	.017	.016	.001	.010	.002

Figure 3 Limiting State Probabilities for the Clusters

	1	2	3	4	5	6	7	8	9	10	11
1	.00	.27	.26	.29	.14	.08	.25	.23	.24	.19	.15
	1.00	.68	.75	.66	.93	.98	.74	.79	.77	.87	.95
2	.27	.00	.18	.07	.17	.25	.10	.16	.09	.22	.39
	.68	1.00	.85	.98	.85	.73	.95	.87	.96	.81	.52
3	.26	.18	.00	.15	.22	.24	.23	.16	.13	.17	.33
	.75	.85	1.00	.90	.78	.77	.76	.89	.93	.89	.69
4	.29	.07	.15	.00	.19	.26	.13	.15	.09	.20	.40
	.66	.98	.90	1.00	.82	.72	.91	.89	.96	.85	.51
5	.14	.17	.22	.19	.00	.15	.12	.18	.15	.21	.26
	.93	.85	.78	.82	1.00	.91	.92	.84	.88	.83	.82
6	.08	.25	.24	.26	.15	.00	.23	.18	.23	.15	.21
	.98	.73	.77	.72	.91	1.00	.77	.87	.78	.92	.88
7	.25	.10	.23	.13	.12	.23	.00	.17	.14	.24	.37
	.74	.95	.76	.91	.92	.77	1.00	.85	.89	.76	.57
8	.23	.16	.16	.15	.18	.18	.17	.00	.16	.15	.34
	.79	.87	.89	.89	.84	.87	.85	1.00	.87	.92	.66
9	.24	.09	.13	.09	.15	.23	.14	.16	.00	.20	.34
	.77	.96	.93	.96	.88	.78	.89	.87	1.00	.84	.65
10	.19	.22	.17	.20	.21	.15	.24	.15	.20	.00	.29
	.87	.81	.89	.85	.83	.92	.76	.92	.84	1.00	.77
11	.15	.39	.33	.40	.26	.21	.37	.34	.34	.29	.00
	.95	.52	.69	.51	.82	.88	.57	.66	.65	.77	1.00

Figure 4 Difference Measures for the Limiting State Probability vectors

the value of the direction cosine is displayed under it. For example, the distance between the limiting state probability vector for clusters 3 and 1 is 0.26 while the direction cosine value if 0.75. According to these measures the limiting state probability vectors for clusters 2 and 4 are most similar and those for clusters 4 and 11 are most dissimilar.

From the information available from the log tapes we also measured the holding times for each state in terms of SUP's. The mean value of SUP's used by jobs from clusters in each of their states is shown in Figure 5. A significant degree of variability exists in the average SUP's used for a state when this state is reached by jobs from different clusters. For example a FORTRAN compile by a job from cluster 11 takes an average of 1.8 SUP seconds while the average for a job from cluster 4 is 9.1 SUP seconds.

The effect of the holding times may be taken into account by weighting each element of the limiting state probability vectors by the expected value of the holding time in terms of SUP's. Note that these weighted vectors now actually describe a semi-Markov process. Again we may use our measures of vector differences for these weighted vectors. The Euclidean distance as well as the direction cosines for the weighted vectors are shown in Figure 6. For the weighted vectors the characteristics of clusters 1 and 5 are most similar and those for clusters 10 and 11 most dissimilar. A comparison with Figure 4 indicates that the reason the relative values change for the weighted vectors is due to the differences in the average number of SUP's used for that state.

4. Concluding Remarks

In this paper we explored the use of Markov models to represent the sequence of programs or job steps which constitute a job in the workload of a computer system. The population of jobs was first divided into several clusters and a separate Markov model was created for jobs from each cluster. A Markovian model is a very succinct description of a sequence and therefore is highly valuable as a generative model.

In the study reported here the states for this Markov model were selected on the basis of their frequency of use. A more compact model may be obtained by reducing the number of states by combining several of them into a common state.

The data requirements for reasonable estimates of the transition probability matrices increase as the square of the number of states. Therefore, for some of the smaller clusters it may become very difficult to get meaningful estimates. While the feasibility of a Markov model of a job has been shown, the states to be used for such models is a topic for further investigations.

References

[1] Agrawala, A.K., Mohr, J.M., and Bryant, R.M., "An Approach to the Workload Characterization Problem", Computer, June, pp. 18-32 (1976).

[2] Agrawala, A.K., and Mohr, J.M., "The Relationship between the Pattern Recognition Problem and the Workload Characterization Problem", SIGMETRICS CMGVIII, Nov. 29-Dec 2, 1977, Washington, D.C.

[3] Agrawala, A.K. and Mohr, J.M., "Some Results on the Clustering Approach to Workload Modelling", Proc. CPEUG, New Orleans, Oct. 11-14, 1977.

[4] Agrawala, A.K., and Mohr, J.M., "Predicting the Workload of a Computer System", Proc. CPEUG, Nov. 8-12, 1976, pp. 150-162 (1976).

[5] Agrawala, A.K., and Mohr, J.M., "A Comparison of the Workload on Several Computer Systems", CMG IX, Nov. 1978, San Francisco.

[6] Agarwal, Bhasker D., "A Model of a Job", M.S. Thesis, Department of Computer Science, University of Maryland, 1976.

[7] Howard, Ronald, Dynamic Probabilistic Systems, Vol. I., John Wiley & Sons, 1971.

[8] Howard, Ronald, Dynamic Probabilistic Systems, Vol. II, John Wiley & Sons, 1971.

[9] Bishop, Fienberg and Holland, "Discrete Multivariate Analysis", M.I.T. Press, 1975.

	START	ASG	FREE	USER	ACCOUN	ASM	CATALO	CTS	ED	ELT	FORTRN	FURPUR	LOAD	MAP	SAVE	SPSS	RESUME	SUSPEN	STATUS	TIME	GROUP1	GROUP2	GROUP3	GROUP4	GROUP
1	.3	.8	.8	5.4	.5	2.8	6.1	4.3	3.3	1.4	2.9	2.8	2.6	6.6	2.3	10.0	5.9	4.5	.1	.7	7.4	1.0	8.3	3.3	5.8
2	.2	.0	.8	12.7	.1	9.8	9.7	4.4	3.9	2.8	4.2	8.4	2.7	14.1	4.0	14.8	8.9	4.8	.1	.4	4.6	1.3	18.9	12.1	4.8
3	.3	.0	.8	15.8	.2	2.2	5.7	4.1	3.2	1.2	4.1	5.1	2.8	9.7	3.2	13.0	8.4	4.6	.1	2.8	4.8	4.8	11.8	23.6	5.9
4	.2	.8	.8	73.5	.2	59.7	9.5	4.3	5.8	2.8	9.1	16.9	2.1	29.2	5.6	13.1	9.6	4.9	.1	.4	7.5	22.0	33.9	31.0	4.8
5	.2	.0	.0	5.8	.3	2.8	11.6	4.5	3.2	1.8	3.1	3.3	2.6	6.6	2.4	10.7	7.3	4.7	.1	.6	5.6	1.8	6.2	5.1	2.9
6	.2	.0	.0	188.8	.1	.8	3.5	4.2	6.8	1.4	4.9	3.8	5.6	18.4	2.8	61.3	8.8	4.7	.1	.8	91.0	3.6	212.5	11.9	44.4
7	.2	.0	.0	21.4	.2	22.9	8.2	4.5	3.5	2.1	3.6	3.7	3.0	11.6	2.8	19.8	9.2	4.8	.1	.4	5.7	1.8	22.1	7.9	18.2
8	.1	.0	.0	68.6	.1	12.8	3.8	4.0	2.9	.8	3.3	5.8	3.8	20.5	3.0	54.8	11.7	4.6	.1	.6	6.6	7.8	54.6	1.9	25.0
9	.2	.0	.0	8.5	.2	3.2	6.4	4.5	3.3	2.0	3.1	8.1	2.7	7.1	4.2	9.7	7.5	4.7	.1	.5	4.1	1.5	8.8	9.6	3.2
10	.4	.8	.0	313.5	.1	52.7	3.2	3.9	91.3	.6	3.4	28.8	3.8	17.4	.0	227.6	14.7	4.1	.1	.9	14.8	5.9	283.2	141.2	7.4
11	.2	.8	.0	4.8	.5	2.8	3.8	4.8	2.8	1.3	1.8	1.8	1.9	2.4	2.1	2.0	4.6	4.7	.1	.8	2.2	.7	1.7	1.4	1.8

Figure 5 Holding Time in SUP's

	1	2	3	4	5	6	7	8	9	10	11
1	.00	1.68	1.69	9.70	.38	19.61	1.98	8.74	1.45	57.35	.46
	1.00	.74	.64	.71	.87	.69	.82	.73	.51	.67	.65
2	1.68	.00	.91	8.32	1.49	18.57	1.24	7.66	.98	56.20	1.96
	.74	1.00	.90	.89	.83	.72	.86	.78	.88	.76	.32
3	1.69	.91	.00	8.59	1.60	18.84	1.74	7.96	.94	56.41	1.90
	.64	.90	1.00	.80	.66	.62	.71	.66	.88	.67	.33
4	9.70	8.32	8.59	.00	9.58	11.26	7.86	2.86	9.13	48.15	9.96
	.71	.89	.80	1.00	.70	.93	.94	.96	.64	.96	.31
5	.38	1.49	1.60	9.58	.00	19.56	1.84	8.68	1.31	57.27	.66
	.87	.83	.66	.70	1.00	.60	.83	.64	.64	.62	.52
6	19.61	18.57	18.84	11.26	19.56	.00	17.81	11.14	19.46	38.13	19.86
	.69	.72	.62	.93	.60	1.00	.92	.98	.36	.99	.30
7	1.98	1.24	1.74	7.86	1.84	17.81	.00	6.92	2.08	55.50	2.30
	.82	.86	.71	.94	.83	.92	1.00	.94	.53	.93	.35
8	8.74	7.66	7.96	2.86	8.68	11.14	6.92	.00	8.55	48.74	9.01
	.73	.78	.66	.96	.64	.98	.94	1.00	.44	.99	.31
9	1.45	.98	.94	9.13	1.31	19.46	2.08	8.55	.00	57.05	1.60
	.51	.88	.88	.64	.64	.36	.53	.44	1.00	.43	.27
10	57.35	56.20	56.41	48.15	57.27	38.13	55.50	48.74	57.05	.00	57.59
	.67	.76	.67	.96	.62	.99	.93	.99	.43	1.00	.30
11	.46	1.96	1.90	9.96	.66	19.86	2.30	9.01	1.60	57.59	.00
	.65	.32	.33	.31	.52	.30	.35	.31	.27	.30	1.00

Figure 6 Difference Measures for the Weighted Limiting State Probability Vectors

CASE STUDY IN CAPACITY PLANNING: ANALYSIS OF AN AUTOMATED BIBLIOGRAPHIC RETRIEVAL SYSTEM

R.P. Goldberg, A.I. Levy,
H.S. Schwenk, Jr.

BGS Systems, Inc.
P.O. Box 128
Lincoln, MA 01773

ABSTRACT

This paper discusses a computer system capacity planning analysis of an automated bibliographic retrieval system. A major focus of the study was to determine the computer system requirements necessary to support anticipated bibliographic workloads (through 1983). The paper addresses the methodology used in obtaining performance data and the use of an interactive capacity planning tool, BEST/1^{TM}, to assimilate the data and predict the performance impact of the anticipated workloads.

KEY WORDS

BALLOTS; BEST/1^{TM}; bibliographic retrieval system; capacity planning; computer performance analysis; computer system modeling; library automation; response time; throughput.

1. Introduction

BGS Systems recently performed an analysis of several different automated bibliographic retrieval systems [1][1]. One aspect of the study was to determine the computer system requirements necessary to support the client's bibliographic workload in both the short term (July 1978) and the long term (1983).

The first step in the performance study was to define and agree upon a generic bibliographic cataloging activity which each of the bidder's system had to execute. Next, each bidder was required to use the consensus cataloging activity as a guideline in preparing computer resource consumption values for BGS Systems' use in constructing performance models. One bidder provided measurements based on performance data that had been collected during routine performance monitoring operations. A second bidder performed a series of controlled experiments to extract the data. A significant effort was devoted, on the part of BGS Systems, to ensure that the data received by each institution represented, as closely as possible, the consensus cataloging activity.

The data obtained from each bidder was verified and reformulated, where necessary, to insure the similarity of the cataloging processing activity across the bidders. In all cases, the data was verified and agreed upon by the institutions before being incorporated into performance models. A BGS Systems' proprietary software package, BEST/1 [2], was used to assimilate the data and predict the performance impact of the anticipated

[1] Figures in brackets indicate the literature references at the end of this paper.

client's workload on each of the systems.

In this paper, we will briefly introduce the consensus bibliographic cataloging activity that was chosen. Then we will illustrate the study approach by focusing exclusively on the Stanford University BALLOTS System, *one* of the several sites investigated. For this site, we will briefly summarize the relevant hardware and software configuration information. We will focus on the performance analysis in considerable detail.

2. The Generic Cataloging Activity

A generic activity, described in Figure 1 was agreed upon as a basis for representing computer resource consumption attributable to cataloging operations. Figure 1 represents the various processing paths through a bibliographic system, referred to as title destinies, that an individual title might ultimately take. The two major destinies agreed upon (highlighted in the figure) represent operations for *copy* and *original* cataloging. These two activities, which support the process of title classification, provided the fundamental focus of the analysis[2]. In the copy cataloging case, the path through the figure is interpreted to mean that a title, upon entering the system, is searched against the bibliographic data base and is found to exist in an adequate enough form to permit a copy cataloging process to occur. In the original cataloging case, the data base search does not retrieve the title; rather it must be entered into the system (in an adequate form) through an original cataloging process.

It is important to note that the consensus cataloging activity represented by Figure 1 portrays the ultimate *destiny* of titles through a bibliographic system and not their processing *flow*. A title's actual processing flow is depicted by Figure 2. In this case, in addition to the destiny, the time element involved in the cataloging process is portrayed. Figure 1 assumes that a title, once entered into the flow, completes processing without the possibility of time delays and "loops". This simplification, agreed to by the proposing institutions, was included in the performance analysis to avoid unnecessary complexity.

2 Original cataloging refers to a multitude of activities involved in the process of entering a title in a bibliographic catalog for the first time. This differs from copy cataloging which includes utilizing a previously developed catalog entry to support the process.

3. BALLOTS System Design

The BALLOTS Library Automation System [3] offers on-line service in a multi-library environment to over one hundred and thirty library institutions. A brief overview of BALLOTS hardware and software is presented below.

3.1 BALLOTS Hardware Configuration

The Stanford University BALLOTS computer hardware environment is a 4 million byte IBM 370/168 Model 3 central processor with IBM 2301 drums for virtual storage and operating system residence, IBM 3350 and 3330-11 direct access disk devices for data base storage and a host of magnetic tape drives and unit record peripherals. This configuration supports a wide variety of the University's data processing - BALLOTS contributing to only a small proportion of the total workload.

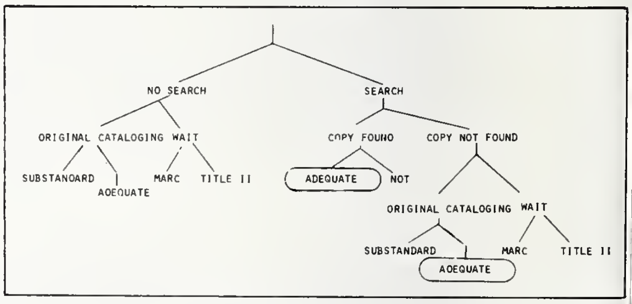


FIGURE 1
CONSENSUS GENERIC CATALOGING ACTIVITY

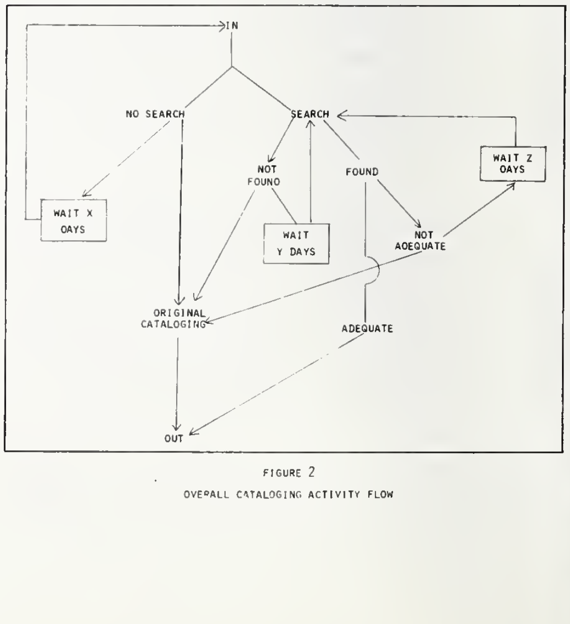


FIGURE 2
OVERALL CATALOGING ACTIVITY FLOW

3.2 BALLOTS Software Environment

BALLOTS was designed to operate within the hardware and software environment depicted in Figure 3. The major system software components of this environment, MILTEN and ORVYL [4], are locally developed and supported. MILTEN provides BALLOTS with teleprocessing control functions supporting both device-dependent and generalized front-end (GFE) communications. The former component provides support for the ZENTEC terminals and the PDP-11/40 interface while the latter component supports the development of specialized device drivers.

Time-sharing services on the Stanford system are provided through the ORVYL time-sharing monitor. ORVYL provides BALLOTS with a flexible time-sliced virtual memory environment and, in addition, provides a sophisticated and security conscious file system. BALLOTS executes as a reentrant subprocessor under ORVYL which in turn, executes within OS/VS2 SVS as an ordinary batch region. All operating system requests (including paging, input/output services, etc.) are handled, independently of the operating system, by invocation of an ORVYL interface routine which issues appropriate supervisor calls to ORVYL for servicing. This design permits BALLOTS to achieve a highly efficient processing potential.

There has also been an attempt, in the design of the system, to optimize the use of main memory. First, the BALLOTS subprocessor code is completely shareable; only one copy is needed regardless of the number of users. Additionally, work space for each user is allocated only as needed and is released back to ORVYL as soon as possible. Since ORVYL runs in a virtual memory environment, memory released in this manner is immediately available for subsequent use.

The BALLOTS subprocessor contains a substantial amount of software extracted from the Stanford Public Information Retrieval System (SPIRES). This code is used for searching the data base files and for updating and decoding records. Much of the software has been modified to provide for the unique needs of BALLOTS. The ORVYL interface mentioned above, and procedures to manage the updating of records and access to individual data elements within a record retrieved by SPIRES each provide examples of modifications included.

In addition to the on-line component, BALLOTS includes a batch component which is invoked nightly to process requests pending from the on-line component. All transactions processed by the on-line component of BALLOTS are queued in a temporary data set for subsequent processing by the batch component. This data set, referred to as the deferred queue, includes updates and additions to the files and requests for the production of library output. Nightly, the BALLOTS on-line component is deactivated and the batch component invoked to service the deferred queue.

3.3 BALLOTS Files

The BALLOTS system supports several on-line files and indexes to them (Figure 4).

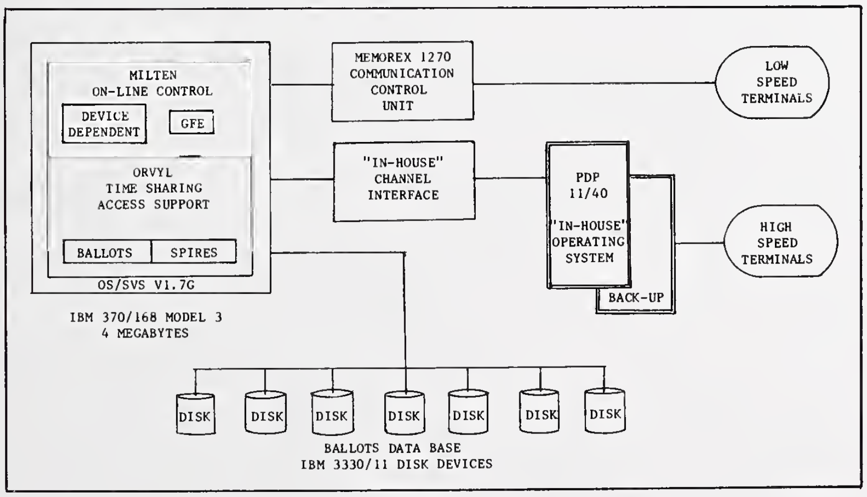


FIGURE 3

BALLOTS ON-LINE HARDWARE/SOFTWARE ENVIRONMENT

The major source files are the MARC, In-Process, Catalog Data and Reference Files[3]. The MARC file contains a complete set of bibliographic records received from the Library of Congress MARC tapes (since 1972) and converted to an internal BALLOTS record format. The In-Process File contains bibliographic and acquisition or in-process control information for items on order or in-process in the Stanford libraries' technical processing division. The Catalog Data File contains bibliographic and holdings information belonging to Stanford libraries and other institutions contributing towards shared cataloging in the BALLOTS data base. The Reference File contains cross references and explanatory notes to the Catalog Data File belonging to Stanford. Records in each of these files are fully accesible to all users.

Two additional files, the Standing Search Request File and the Deferred Queue are also maintained on-line. The former provides a facility for standing search requests to be processed against the MARC File[4]. The latter contains the daily queue of cataloging transactions for BALLOTS overnight processing.

3 The In-Process, Catalog Data and Reference Files are physically maintained as one SPIRES file, but are logically considered to be three separate files.

4 A standing search request is a prestored machine readable request to "automatically" search incoming LC MARC records in an attempt to locate a particular one for purposes of copy cataloging.

Indexes to each of the files described above are identified in Figure 4. BALLOTS uses word and phrase indexes in addiiton to the typical library file indexes: Library of Congress Card Number (LCCN), and Call Number. For word indexes, all significant words in the value of an indexed data element are indexed with frequently occurring words being omitted. For phrase indexes, of which only topical and geographical subjects are included, an entire subject heading is treated as a single index term.

BALLOTS files are physically maintained through SPIRES [5] (in the ORVYL file system) in two distinctly different types of structures. The first, incorporated into the Standing Search Request, In-Process, Catalog Data and Cross Reference files, is the SPIRES slot record structure. The second, incorporated into the Active MARC File, Deferred Queue and all index files, is the SPIRES B-tree organization.

The SPIRES slot record structure provides a relatively simple file organization. Records are maintained through an essentially fixed length element array format based on a system assigned monotonically increasing key (the BALLOTS Local Identification Number). The records are stored in a sequentially organized residual file with the slot structure housing each record's key and residual address.

The B-tree (or balanced tree) organization is slightly more complex. This organization can best be described as a tree structure with between k+1 and 2k+1 son nodes for any given father node. Individual records in the tree

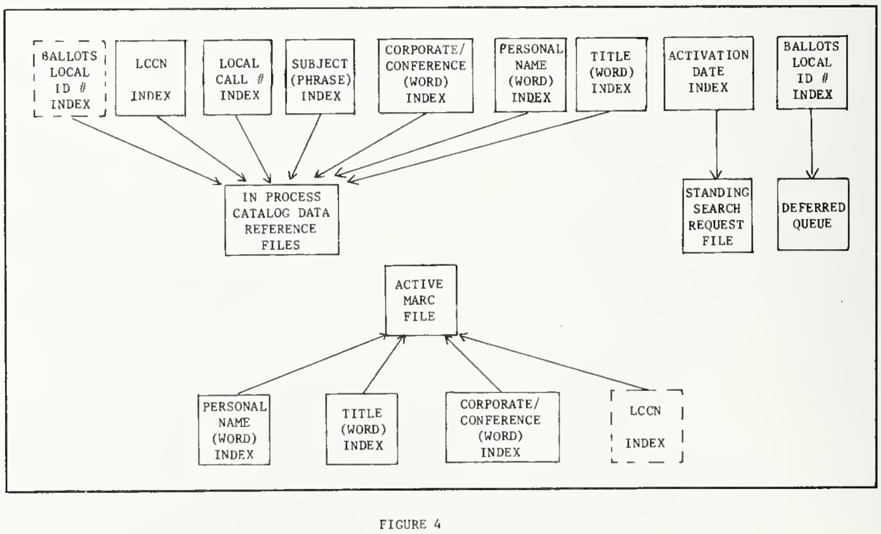


FIGURE 4

BALLOTS ON-LINE BIBLIOGRAPHIC FILE ORGANIZATION are organized so that postorder tree traversal retrieves records in their key collating sequence.

In addition, Stanford is currently developing a substantial file system enhancement called the network file design. Issues currently within the scope of network file design include:

* Enhanced index files
* Hashed index base block organization
* Multi-level deferred queue organization
* Authority control development
* Storage schemes for multi-institutional data

4. BALLOTS System Performance

4.1 BALLOTS Interpretation of the Generic Cataloging Activity

Stanford University personnel provided an interpretation of how the generic cataloging activities would be performed using BALLOTS. Figure 5 illustrates the transactions invoked in original cataloging and Figure 6 illustrates the transactions invoked in copy cataloging ("copy adequate").

4.1.1 Original Cataloging

A "typical" BALLOTS original cataloging activity (Figure 5) consists of the following series of interactions:

FIND CARD# - An attempt is made to locate the subject item by Library of Congress Card Number. Since this scenario is original cataloging, the *FIND* is presumed to fail.

FIND OTHER - An Attempt is made to locate the subject item by using one or more of the BALLOTS indexes. The terminal operator can be counted on to issue the command that is the easiest to enter, and yet has the desired precision.

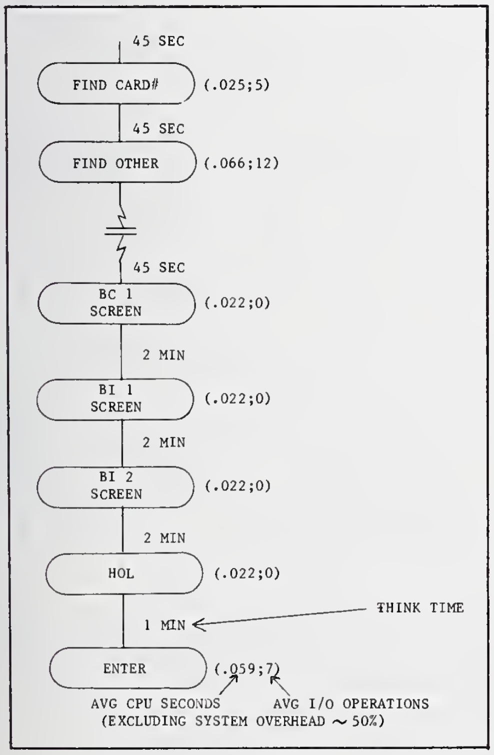


FIGURE 5

BALLOTS ORIGINAL CATALOGING ACTIVITY

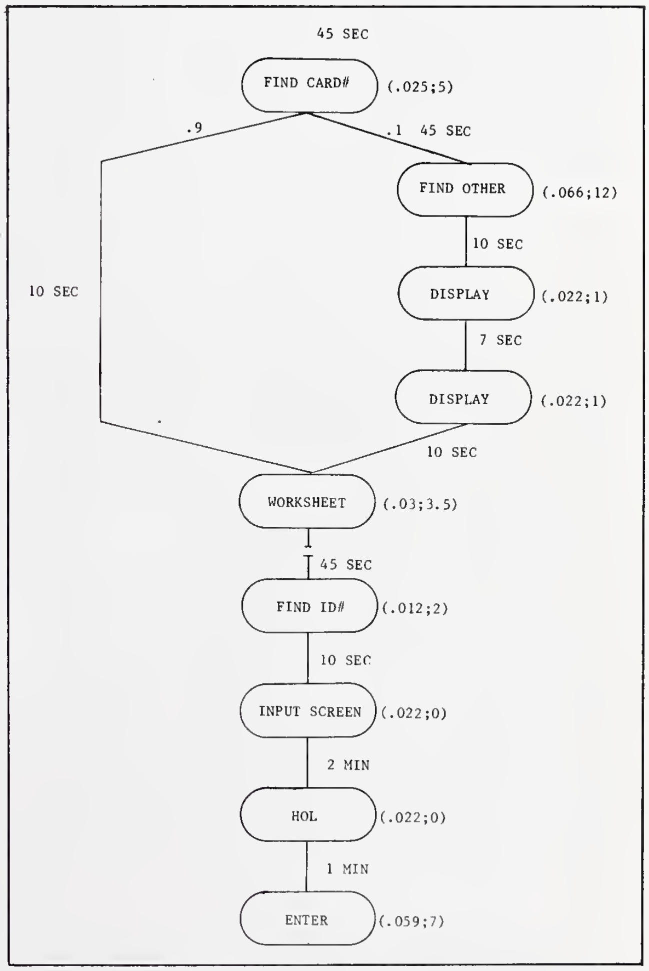


FIGURE 6

BALLOTS COPY CATALOGING ACTIVITY

At this point it is determined that an original cataloging activity must be undertaken. The information for this activity must be obtained. When the information is available (some time later), the terminal operator may continue processing the subject item.

BC1 Format - This format is used to record miscellaneous control information about the item being cataloged that does not constitute "bibliographic description" per se. All of the data element mnemonics possible on the BC1 format are supplied by the system; the user needs only to fill in the data, which in many cases consists of a brief code.

BI1 Format - This format is used to record the bibliographic description, (main entry through size) of the item being cataloged. All of the data element mnemonics possible on the BI1 format are supplied by the system.

BI2 Format - This format is used to record the remainder of the bibliographic description from series notes through tracing, as well as to indicate any additional catalog cards to be printed and their specific headings. No element mnemonics are prompted on this format by the system; the user keys both the desired mnemonics and the information following them.

HOL Format - This format is used to record the call number, shelving location, copy and volume numbers for an item being cataloged, and to instruct the system to print catalog cards and/or magnetic tape output. This is the final (required) format in the cataloging sequence. It contains a combination of input fields labeled with their data element mnemonics by the system and input fields into which the user keys both the mnemonics and the value of data elements wanted in the record.

ENTER - Enter the title into the appropriate catalog data file and create all necessary index entries. Since the design does not update these files during interactive use, the predominant work executed on behalf of this command is actually delayed until batch update processing later in the day (see below). Consequently, the primary work performed by this command, during daytime, interactive use, is the establishment of a deferred queue update request element for later processing.

4.1.2 Copy Cataloging

A "typical" BALLOTS copy cataloging activity (Figure 6) consists of the following series of interactions:

FIND CARD# - An attempt is made to locate the subject item by Library of Congress Card Number. Since this scenario is copy cataloging, the *FIND* will *very often* be successful. If it is successful, the terminal proceeds to requesting worksheet preparation. If it fails, a *FIND OTHER* is issued.

FIND OTHER - An attempt is made to locate the subject item by using one or more of the BALLOTS indexes (see original cataloging, above). Since this is the scenario for copy cataloging, any item that has not been found via LCCN will be found using this request.

DISPLAY (twice) - A successful *FIND OTHER* locates several candidate items. On the average, past statistics indicate that two of them must be displayed to uniquely identify the item of interest.

WORKSHEET - Production of a worksheet is requested. The actual printing of the worksheet is performed as a batch activity.

At this point, the copy cataloging activity is underway. The worksheet includes information to be verified as well as new information to be obtained. When the information is available (some time later), the terminal operator may continue processing the subject item.

FIND ID# - The existing entry, which is the subject of copy cataloging, is (efficiently) located using a unique identifier which was printed on the worksheet.

INPUT SCREEN - One of the input screen formats (e.g., BC1, BI1, BI2) is used to add or modify information. (See original cataloging, above).

HOL Format - See orginal cataloging, above.

ENTER - See original cataloging, above.

4.1.3 Overnight Update and Worksheet Preparation

The "typical" BALLOTS original cataloging and copy cataloging activities generate some batch processing work to be performed. The batch processing is not explicitly requested by the terminal operator but rather is implicitly produced by BALLOTS interactive components.

Original cataloging generates a batch request to enter the new bibliographic item into the appropriate catalog data file and update the BALLOTS indexes accordingly. In addition to this activity, copy cataloging will have produced (sometime earlier) a cataloging worksheet.

Neither scenario includes catalog card preparation.

4.1.4 Measurement Environment

Stanford University personnel chose to provide performance data taken from the actual BALLOTS production environment. Special experiments of controlled usage of BALLOTS were not necessary since there already existed a substantial amount of performance management information available. This information was available both as extensive raw data and in management summary form.

The measurement environment was the Stanford University IBM 370/168 computing facility described previously. The software included OS/VS2 SVS, WYLBUR, ORVYL, MILTEN, SPIRES, and BALLOTS. While the (interactive) BALLOTS measurements were being taken, a variety of other jobs, completely independent of BALLOTS (ORVYL, WYLBUR and batch processing), were executing concurrently.

4.2 BALLOTS Resource Consumption

Stanford University personnel provided both summarized and raw resource usage measurement for several different aspects of the BALLOTS system:

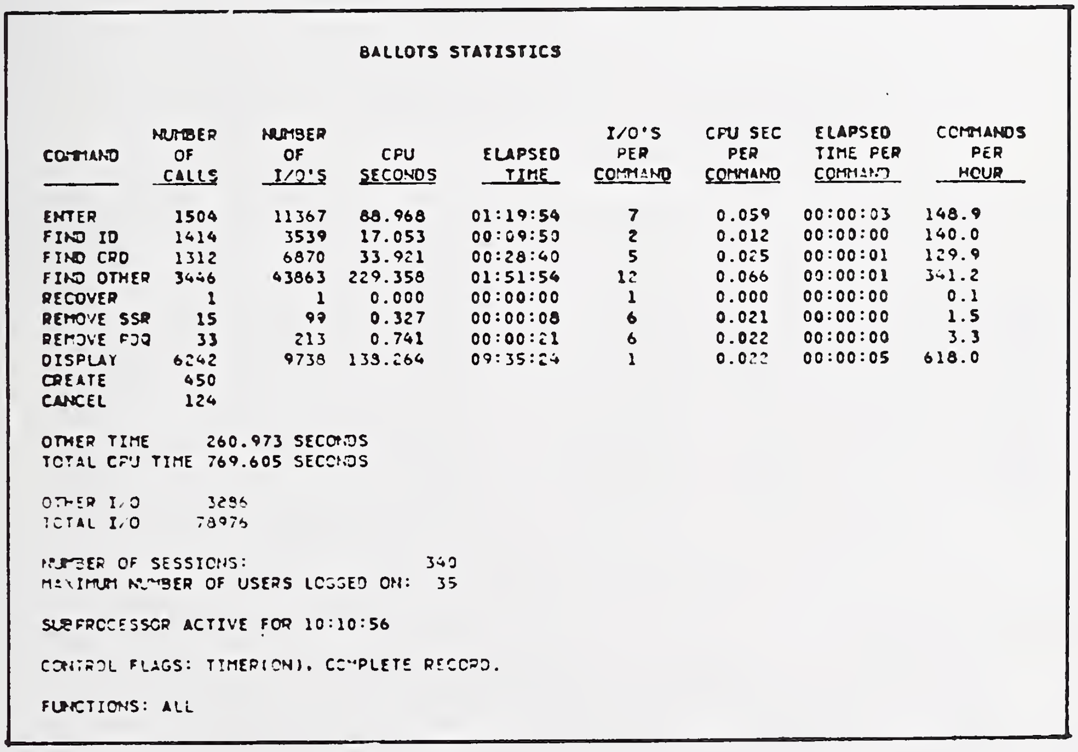
* BALLOTS Interactive Commands
* BALLOTS Batch Processing
* System Overhead
* BALLOTS File Organization

Each of these aspects is discussed below in detail. Finally, the data is combined to produce a coherent BALLOTS resource usage value.

4.2.1 BALLOTS Interactive Commands

Figure 7 provides the results of measurements of BALLOTS typical resource usage. Thus, for example, during the ten hours of BALLOTS use *FIND OTHER* was called 3446 different times causing an average of 12 I/O oeprations and .066 seconds of CPU time per use. These values do *not* include system overhead.

Stanford University personnel then applied these usage values to the commands employed in the typical original cataloging and copy cataloging scenarios presented above. Referring back to Figures 5 and 6, we see the CPU time and I/O operation count per command. These values are based on Stanford Personnel's review of a number of different sets of raw data. Once again, these values do not include system overhead.



```
                              BALLOTS STATISTICS

             NUMBER    NUMBER                             I/O'S    CPU SEC   ELAPSED    COMMANDS
COMMAND        OF        OF      CPU        ELAPSED        PER       PER     TIME PER     PER
             CALLS     I/O'S   SECONDS        TIME       COMMAND   COMMAND   COMMAND      HOUR

ENTER         1504     11367    88.968     01:19:54         7       0.059    00:00:03   148.9
FIND ID       1414      3539    17.053     00:09:50         2       0.012    00:00:00   140.0
FIND CRD      1312      6870    33.921     00:28:40         5       0.025    00:00:01   129.9
FIND OTHER    3446     43863   229.358     01:51:54        12       0.066    00:00:01   341.2
RECOVER          1         1     0.000     00:00:00         1       0.000    00:00:00     0.1
REMOVE SSR      15        99     0.327     00:00:08         6       0.021    00:00:00     1.5
REMOVE FDQ      33       213     0.741     00:00:21         6       0.022    00:00:00     3.3
DISPLAY       6242      9738   138.264     09:35:24         1       0.022    00:00:05   618.0
CREATE         450
CANCEL         124

OTHER TIME      260.973 SECONDS
TOTAL CPU TIME  769.605 SECONDS

OTHER I/O        3296
TOTAL I/O       78976

NUMBER OF SESSIONS:                  340
MAXIMUM NUMBER OF USERS LOGGED ON:    35

SUBPROCESSOR ACTIVE FOR 10:10:56

CONTROL FLAGS: TIMER(ON), COMPLETE RECORD.

FUNCTIONS: ALL
```

FIGURE 7

BALLOTS RESOURCE UTILIZATION

4.2.2 BALLOTS Batch Processing

Figure 8 provides the result of direct measurements of BALLOTS batch processing. Stanford University personnel judged this batch to be typical.

Referring to Figure 8, we see that 1815 titles resulted in adds/modifies to the data base (1001 + 814 = 1815). These were processed via the overnight update program FRMSUAØ3 which required 11.827 minutes of CPU time and utilized 109,012 I/O operations. This processing corresponds to .390 seconds of CPU time and 60.06 I/O operations per title. As before, the CPU value does not include system overhead.

However, it does include several components which generate considerable resource usage[5]. Sufficient checkpointing is done by BALLOTS to insure restorability without human intervention should the run be interrupted due to hardware or software failure. Also included in these statistics is the activity which accrues from the dynamic local rebalancing of trees. BALLOTS intention is to reorganize the data base in process rather than to perform periodic massive reorganization.

Again referring to Figure 8, we see that program FRMSUAØ8 (catalog data slips, i.e., worksheets) consumed .150 minutes of CPU time and 2080 I/O operations. Stanford personnel were able to estimate that this corresponded to approximately 1000 reuqests of .009 seconds of CPU time and (approximately) 2 I/O operations per worksheet. The CPU value does not include system overhead.

5. Some of this resource usage has been reduced by implementing system modifications which were identified during this study. See section "Modified BALLOTS Resource Consumption".

BALLOTS CENTER

Batch Statistics

Acquisitions (IPF Records Added/Modified)	814
Titles Catalogued (CDF REcords Added)	1,001
Catalog Cards Produced	5,811
Standing Search Requests Added	105

Program ID	Wall Time	I/O's	CPU Min.	Lines Printed
FRMSUA01 (DEFQ Archive)	00:02:49	3,013	.10	295
FRMSUA02 (Formatting)	00:07:32	5,415	1.671	2,320
FRMSUA03 (Overnite Update)	01:10:50	109,012	11.827	270
FRMSUA04 (Stanford Cards)	00:02:21	2,027	.100	22,452
FRMSUA05 (Purchase Orders)	00:01:00	662	.040	6,242
FRMSUA06 (Spine Labels)	00:00:31	25	.010	1,757
FRMSUA07 (File Slips)	00:01:14	593	.050	8,111
FRMSUA08 (Cat Data Slips)	00:02:13	2,080	.150	31,669
FRMSUA09 (Batch Network Termination)	00:00:16	-	.010	112

OVERNITE SUMMARY:

Adds/Modifies to REC9: 814 + 1001 = 1815
CPU per Add/Modify: .390 sec.
I/O's per Add/Modify: 60.06

FIGURE 8

BALLOTS BATCH STATISTICS

4.2.3 System Overhead

Figure 9 provides the results of direct measurement of the ORVYL environment during BALLOTS interactive processing. It was asserted by Stanford University personnel that these measured values are typical and stable.

Referring to Figure 9, we see (from the top several lines of the figure) that while 477.031 seconds total CPU time was attributed to ORVYL users (such as BALLOTS), an additional system overhead for OS EXCP time (56.247 seconds) and ORVYL OVERHEAD (194.814) was not. Thus, for all ORVYL users such as BALLOTS the reported ORVYL CPU time must be multiplied by a factor equal to approximately 1 plus (56.247 + 194.814)/477.031 or about 1.5. Stanford University personnel have concurred with this estimate.

```
STATISTICS FROM JOB ORVYL

ELAPSED TIME = 21:33:15
CHARGED TIME=477.031 SEC.,(EXEC:281.393 SEC.,SVC:75.389 SEC.,I/O:120.244 SEC.)  65% }
OS EXCP TIME=56.247 SEC.,(PAGE:34.300 SEC.,FILE:21.897 SEC.,KEY:0.048 SEC.)   7%  }    X 10
ORVYL OVERHEAD=194.814 SEC.,(POLL:131.305 SEC.,PFLT:37.811 SEC.)  26%             }
MEMORY CHARGES=107508098 PAGE-SECONDS.

DISPATCHING:   <.125  - .250  - .500  -  1.0  -  2.0  -  4.0  -  8.0  - >16.0
PAGE FAULT    427705   43645   35328   26104   14977    6818    2554     651
USER WAIT     864840  272231  358962  138794   70066   22386    8213    3604
INTERRUPT      57361   20857   31013   39436   38332   34436   19920    7304
TIME SLICE     15930    7523   11067   15609    9012    7933    5052    4295

PAGING: 256 PAGE FRAMES, 3229 EXTERNAL PAGES (USE: 337 CUPR, 2239 MAX)
DEN   EXCPS        PAGES      AVG:1.26
1 703241  79%  703241  62%
2 133013  15%  266026  23%
3  36034   4%  108102   9%
4  10156   1%   40624   3%

34638 BLOCKS AVAILABLE, 736230 TOTAL
62846 DATA WRITES, 355525 DATA READS   23700 IXR WRITES, 58397 IXR READS
735768 I RS REQUESTED, 677371 ALREADY IN VIRTUAL MEMORY  92.06%
41332 IXR SLOTS STOLEN, 6489 REQUIRED WRITE  15%
TOTAL FILE I/O'S: 523468, MAXIMUM CONCURRENT FILES ATTACHED: 460

I/O QUEUE: TYPE REQUESTS IN USE  WAITS
           FILE   495808      8      0
           PATH   255599      2      0
           KEY      2664      2      0

DRUM  PAGES      AVG. TIME (MS)
  0      67754         50
  1     629674         13
  2     385134         14
  3      35431         21

DISK  I/O COUNT  AVG. TIME (MS)
  0      84424         37
  1       9232         35
  2      28152         36
  3      46764         36
  4      50593         39
  5      56236         34
  6     126091         31
  7     121986         33

KEY   I/O COUNT  AVG. TIME (MS)
  0       1327        100
  1          5        126

ORVYL WAIT: 1604304  6919.265 SEC.  11182.151 SEC.;  BATCH SCB: 1601822  6901.232 SEC.  11182.136 SEC.
POLLED 4227595 TIMES, SEARCHED 1453243 USERS
MAXIMUM RUN QUEUE SIZE: 12

PATH TRANSACTION STATISTICS
                       COUNT    TOTAL BYTES   AVG.LENGTH
  MILTEN
    XACTS SENT         292956     27523723          93
```

FIGURE 9

ORVYL JOB STATISTICS

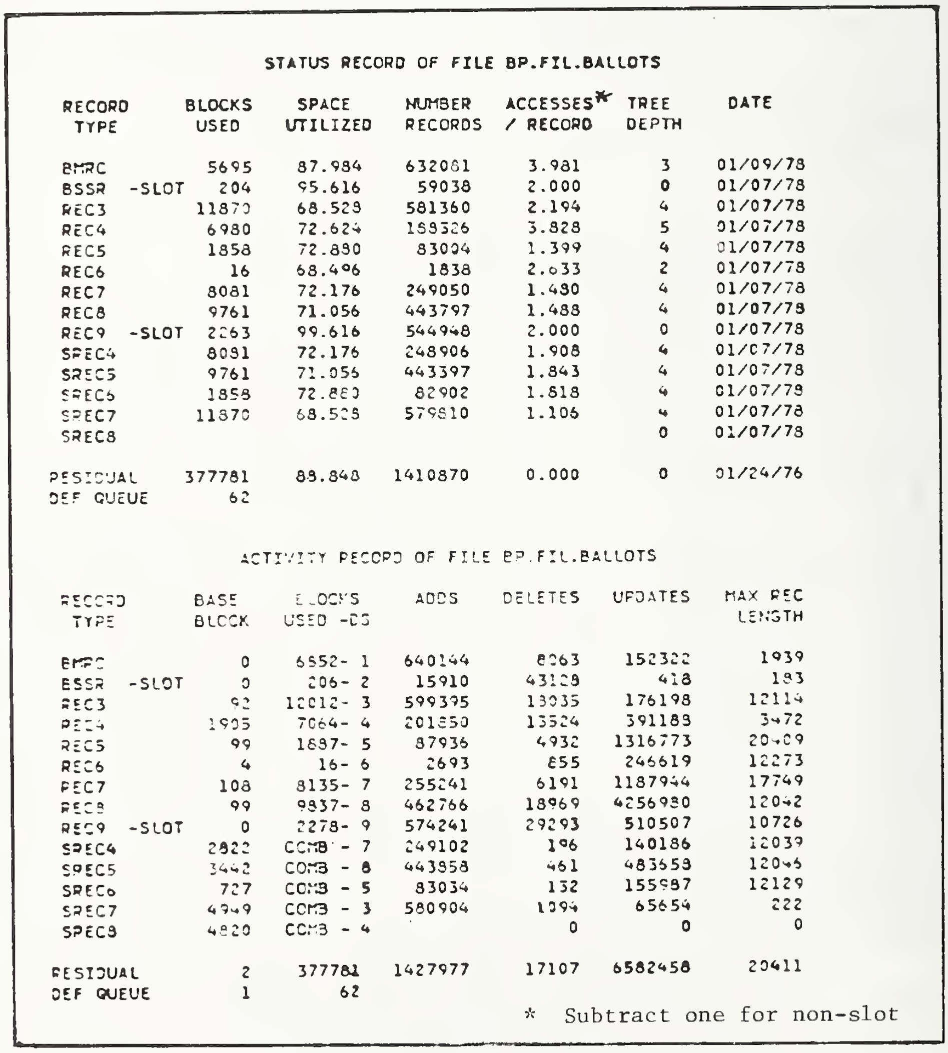
4.2.4 BALLOTS File Organization

Figure 10 provides the results of direct measurements on the files which make up the BALLOTS data base. It is the file organization and size which affects the number of I/O operations per BALLOTS command and batch update. Referring to the BMRC record type of Figure 10, we see that an average of 3.981 accesses per record is required for the 632,081 records of the MARC file. Stanford personnel have asserted that the number of accesses is a parameter whose value has increased very slightly in several years. This is not surprising with the B-tree file structure.

A first order analysis shows that the average degree of branching for a tree of depth 3.981 is 28.65 per level. Even if the MARC file increases by an order of magnitude to 6,000,000 records, the number of accesses will just increase to approximately 4.65. Thus, a 900% increase in the size of the MARC file causes a 17% increase in the number of accesses. Since the top level index remains main memory resident, the number of I/O operations might increase by approximately 22% when the MARC file grows 900%.

4.2.5 BALLOTS Directly Measured Resource Consumption

Referring back to the usage values reported in Figure 5, we see that the interactive component of original cataloging requires a total of 238 msec of direct BALLOTS CPU time and a total of 24 I/O operations. Since the capture ratio is 1.5 for both the interactive and batch workload, the total CPU time (including overhead) for original cataloging is .357 seconds.



STATUS RECORD OF FILE BP.FIL.BALLOTS

RECORD TYPE		BLOCKS USED	SPACE UTILIZED	NUMBER RECORDS	ACCESSES* / RECORD	TREE DEPTH	DATE
BMRC		5695	87.984	632081	3.981	3	01/09/78
BSSR	-SLOT	204	95.616	59038	2.000	0	01/07/78
REC3		11870	68.528	581360	2.194	4	01/07/78
REC4		6980	72.624	158326	3.828	5	01/07/78
REC5		1858	72.880	83004	1.399	4	01/07/78
REC6		16	68.496	1838	2.633	2	01/07/78
REC7		8081	72.176	249050	1.480	4	01/07/78
REC8		9761	71.056	443797	1.488	4	01/07/78
REC9	-SLOT	2263	99.616	544948	2.000	0	01/07/78
SREC4		8081	72.176	248906	1.908	4	01/07/78
SREC5		9761	71.056	443397	1.843	4	01/07/78
SREC6		1858	72.880	82902	1.818	4	01/07/78
SREC7		11870	68.528	579810	1.106	4	01/07/78
SREC8						0	01/07/78
RESIDUAL		377781	88.848	1410870	0.000	0	01/24/76
DEF QUEUE		62					

ACTIVITY RECORD OF FILE BP.FIL.BALLOTS

RECORD TYPE		BASE BLOCK	BLOCKS USED -DS	ADDS	DELETES	UPDATES	MAX REC LENGTH
BMRC		0	6552- 1	640144	8063	152322	1939
BSSR	-SLOT	0	206- 2	15910	43128	418	183
REC3		92	12012- 3	599395	13035	176198	12114
REC4		1905	7064- 4	201850	13524	391189	3472
REC5		99	1887- 5	87936	4932	1316773	20409
REC6		4	16- 6	2693	855	246619	12273
REC7		108	8135- 7	255241	6191	1187944	17749
REC8		99	9837- 8	462766	18969	4256980	12042
REC9	-SLOT	0	2278- 9	574241	29293	510507	10726
SREC4		2822	COMB - 7	249102	196	140186	12039
SREC5		3442	COMB - 8	443858	461	483658	12046
SREC6		727	COMB - 5	83034	132	155987	12129
SREC7		4949	COMB - 3	580904	1094	65654	222
SREC8		4820	COMB - 4		0	0	0
RESIDUAL		2	377781	1427977	17107	6582458	20411
DEF QUEUE		1	62				

* Subtract one for non-slot

FIGURE 10

BALLOTS RESOURCE STATISTICS

Referring back to the usage values reported in Figure 6, we can derive the resource usage for the interactive component of copy cataloging. User behavior measurements on the current BALLOTS system indicate that 90% of the adequate cataloging is the result of a successful *FIND CARD#* command while only 10% is the result of a successful *FIND OTHER* command. If these "branching probabilities" for copy cataloging user behavior are employed, we see that the interactive component of copy cataloging requires a total of 144 msec of direct BALLOTS CPU time and a total of 18 I/O operations. Since the capture ratio is 1.5, the total CPU time (including overhead) for copy cataloging is .216 seconds.

Batch processing resource usage must also be adjusted by the capture ratio. Thus, the batch update CPU usage of .390 seconds becomes .585 seconds. This value was computed as .591 seconds in other sets of measurements and .591 was the value agreed to. In addition, the value of 60.06 I/O operations was adjusted up to 62 I/O operations by reference to other sets of batch statistics.

Batch worksheet preparation was adjusted up to .015 seconds and 2 I/O operations.

This data is summarized in Table 1 below:

TABLE 1

	ORIGINAL CATALOGING		COPY CATALOGING	
	CPU SECONDS	I/O OPERATIONS	CPU SECONDS	I/O OPERATIONS
Interactive	.357	24	.216	18
Batch to enter in Data Base	.591	62	.519	62
Batch Worksheet Preparation	-	-	.015	2
TOTAL	.948	86	.822	82

4.2.6 Modified BALLOTS Resource Consumption

In addition to the directly measured BALLOTS resource usage just presented, two other sets of values were estimated. These values are the anticipated result of both immediate and longer term modifications to BALLOTS.

In the course of the BALLOTS performance analysis, Stanford University personnel identified a short term opportunity to tune the batch file update program. This effort, considered relatively minor could significantly affect batch resource consumption. The tuning is being performed by restructuring the SPIRES overnight processor modules. Currently there are two large modular portions. One part is the file service routines which provide means for accessing and replacing or adding records. The other portion is a general processor which develops indexes from records and performs general utility processing. Adding a pointer to an index record involves reading in the index record, reconstructing it with the pointer added, and then updating the index record. The current implementation utilizes both the file service module and the general processor. Since the general processor is unaware that the index tree has already been traversed by the file service module, an extra and unnecessary tree traversal is performed in the update operation.

By eliminating this extra tree traversal, Stanford personnel believe that batch resource usage will be reduced significantly. The estimated impact could reduce (total) batch CPU time from .591 seconds to approximately .473 seconds and *batch* I/O operations from 62 to approximately 45. This tuning is currently in test and is scheduled to go into production[6].

The "tuned" resource usage is summarized in Table 2.

TABLE 2

	ORIGINAL CATALOGING		COPY CATALOGING	
	CPU SECONDS	I/O OPERATIONS	CPU SECONDS	I/O OPERATIONS
Interactive	.357	24	.216	18
Batch to enter in Data Base	.473	45	.473	45
Batch Worksheet Preparation	-	-	.015	2
TOTAL	.830	69	.704	65

The Network File Design, discussed previously is a longer term system modification predicted to have a dramatic impact on BALLOTS performance. The major performance improvement promises to come from the structure of the Network File Design which eliminates the need for multiple records when two or more libraries catalog the same work. Thus, indexing need not be done in copy cataloging or in cataloging a new edition of a previously cataloged work.

Based upon the physical structure of the Network File Design, the anticipated batch processing requirements, minor changes to

6 The modification has been made and measurements corroborate the estimates.

several of the interactive commands, and a modification of the file structure for deferred queue request, Stanford University personnel estimated and substantiated new resource usage values. These are presented in Table 3.

TABLE 3

	ORIGINAL CATALOGING		COPY CATALOGING	
	CPU SECONDS	I/O OPERATIONS	CPU SECONDS	I/O OPERATIONS
Interactive	.327	21	.186	15
Batch to enter in Data Base	.500	42	.150	14
Batch Worksheet Preparation	-	-	.015	2
TOTAL	.827	63	.351	31

All except one of these estimates is directly relatable to the direct measurements taken on the current system modified to reflect minor changes. The estimate which is most difficult to verify is the CPU and I/O copy cataloging resource consumption for batch entry into the data base, since it is based on a completely new design.

4.3 BALLOTS Response Time Analysis

A BEST/1[7] model was developed in order to predict the BALLOTS central system interactive and batch performance on a particular hardware configuration under a number of different workloads. Stanford University designated a proposed dedicated BALLOTS[8] hardware configuration to be modeled in the BEST/1 performance analysis. The most important components of their proposed central system configuration are:

* Amdahl 470/V-5 central processor - This is compatible with the current IBM 370/168 and is (as per agreement, by Stanford University personnel) approximately the same speed.
* 4 million bytes of main memory
* 12 input/output channels
* IBM 2835 fixed head controller with 2 IBM 2305-2 fixed head storage devices. These devices will provide swap space for BALLOTS users whose work spaces are written out of main memory
* 12 spindles of IBM 3330-11 disks - The BALLOTS data base will use 6 of these disks, i.e., 1.2 billion bytes.

4.3.1 BEST/1 BALLOTS Interactive Model

The BEST/1[9] model for BALLOTS interactive processing required several other input parameters to be specified. These included values to describe the utilization of main memory, the drum swapping behavior, and the file access behavior.

It was assumed that in a dedicated BALLOTS environment, SVS, ORVYL, MILTEN, SPIRES, and BALLOTS would always be main memory resident and work areas associated with terminal operators would be swapped in and out of main memory as needed. SVS, BALLOTS, et al require approximately 1.3 megabytes of main memory. A BALLOTS user work area varies from 20K bytes to 120K bytes. In the absence of detailed measurements of the memory size distribution, the mean value was estimated by Stanford University personnel to be 80K bytes. Thus, in a 2M byte main memory, 9 users can fit into main memory concurrently; in a 3M byte machine 21 users can fit and in a 4M byte machine 34 users can fit.

The drum swapping behavior also had to be specified. Existing BALLOTS user behavior measurements indicated that a terminal operator typically requires at least 45 seconds ("think time") between BALLOTS commands. Under these circumstances, if the system is heavily loaded, all main memory pages associated with this user will have been reassigned. This

7 BEST/1 is a proprietary interactive software package of BGS Systems, Inc.

8 In addition to the cataloging workload described in this analysis, the dedicated hardware configuration was assumed to be capable of supporting non-interfering, lower priority work that could (if desired) utilize some of the systems' residual capacity. The characterization of this low priority work and the consequent performance implications were *not* part of this study.

9 Note that BEST/1 can be utilized to model computer systems much more complex than the one presented here. While the subject system has a single workload (transaction processing), a single CPU, and a total of only eight drums and disks, BEST/1 can model many concurrent workloads (transaction processing, timesharing, batch including multiple different classes of each), multiple CPUs and hundreds of DASDs. In addition, BEST/1 directly supports the priority scheduling disciplines which are common in operating systems such as OS/VS2 MVS.

implies that each interaction will require all user pages to be swapped in. In the absence of detailed measurements, Stanford University personnel estimated that each swap in will require half of the currently resident pages to be swapped out due to modification. Thus, each of the seven interactions of original cataloging will require a total of 30 I/O operations on the 4K-byte pages. It was assumed that pages are routinely distributed across the two drums and the drums are lightly utilized.

Finally, the file accessing behavior had to be specified. It was assumed that all I/O operations are randomly distributed across all spindles, i.e., 6, containing BALLOTS files.

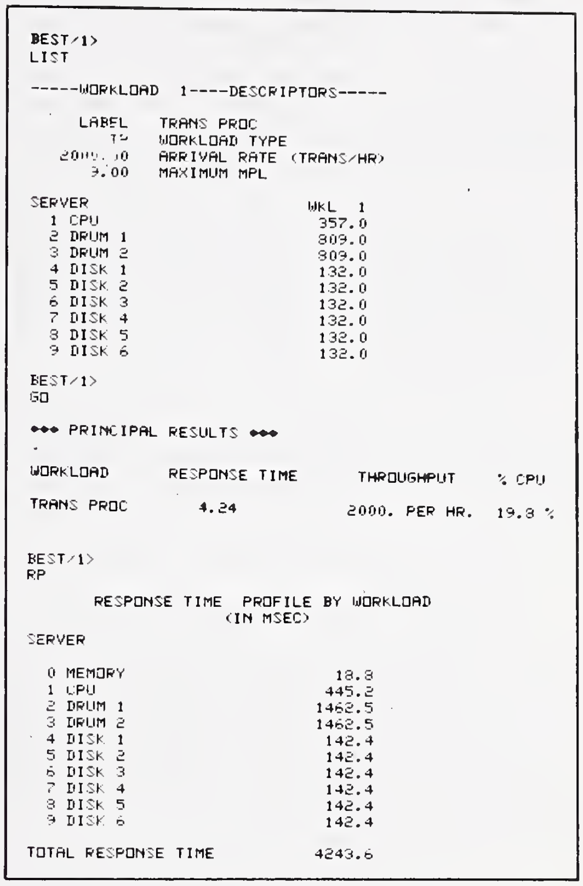
BEST/1 input parameters were then developed from the resource usage values (previously collected), the designated configuration, and the modeled behavior. These parameters for the direct measured current system software for original and copy cataloging are shown in Table 4.

TABLE 4

BALLOTS INTERACTIVE PROCESSING MODEL

DEVICE	ORIGINAL CATALOGING	COPY-ADEQUATE
CPU - V-5	357 msec	216 msec
DRUM1 - 2305	809 msec	728 msec
DRUM2 - 2305	809 msec	728 msec
DISK1 - 3330-11	132 msec	99 msec
DISK2 - 3330-11	132 msec	99 msec
DISK3 - 3330-11	132 msec	99 msec
DISK4 - 3330-11	132 msec	99 msec
DISK5 - 3330-11	132 msec	99 msec
DISK6 - 3330-11	132 msec	99 msec

A set of interactive BEST/1 runs was made in which the transaction arrival rate and the amount of memory was varied. Sample dialog and output from one such run is shown in Figure 11. In this run, BEST/1 modeled a configuration with a 2M byte main memory, e.g., maximum MPL (multiprogramming level) of 9.0, serving titles arriving from terminal operators at the rate of 2000 per hour. Even at this arrival rate, it was determined that 2M byte would provide sufficient main memory and so this value was used for the further analysis. Also, it was observed that the interactive portion of original cataloging consumes more resources than copy cataloging does. Consequently, the performance of BALLOTS handling a 100% original cataloging workload would serve as an upper bound on how a mixed original/copy cataloging workload would actually behave in practice.



```
BEST/1>
LIST

-----WORKLOAD  1----DESCRIPTORS-----

     LABEL   TRANS PROC
        TP   WORKLOAD TYPE
   2000.00   ARRIVAL RATE (TRANS/HR)
      9.00   MAXIMUM MPL

SERVER                          WKL  1
  1 CPU                          357.0
  2 DRUM 1                       809.0
  3 DRUM 2                       809.0
  4 DISK 1                       132.0
  5 DISK 2                       132.0
  6 DISK 3                       132.0
  7 DISK 4                       132.0
  8 DISK 5                       132.0
  9 DISK 6                       132.0

BEST/1>
GO

*** PRINCIPAL RESULTS ***

WORKLOAD       RESPONSE TIME       THROUGHPUT      % CPU

TRANS PROC         4.24          2000. PER HR.    19.8 %

BEST/1>
RP

          RESPONSE TIME  PROFILE BY WORKLOAD
                       (IN MSEC)
SERVER

  0 MEMORY                        18.8
  1 CPU                          445.2
  2 DRUM 1                      1462.5
  3 DRUM 2                      1462.5
  4 DISK 1                       142.4
  5 DISK 2                       142.4
  6 DISK 3                       142.4
  7 DISK 4                       142.4
  8 DISK 5                       142.4
  9 DISK 6                       142.4

TOTAL RESPONSE TIME             4243.6
```

FIGURE 11

SAMPLE BEST/1 MODEL

The results of the BEST/1 model for the interactive portion of original cataloging using parameters based on measured values from the current system are shown in Table 5.

TABLE 5

ARRIVAL RATE (Titles per hour)	% CPU UTILIZATION	TOTAL RESPONSE TIME (Sum of 7 interactions)
500	5.0%	3.0 sec
1000	9.9%	3.3 sec
1500	14.9%	3.69 sec
2000	19.8%	4.24 sec

Since there are approximately 2000 hours per year[10], the client's 1978 workload is 200 titles per hour and the client's 1983 workload is projected to be 400 titles per hour. Thus, even by 1983, the client will consume less than 5% of the prime shift CPU. Original cataloging total response time from the central system will be less than 3 seconds with an average response time of less than 0.43 seconds for each of the 7 interactions.

If other non-client BALLOTS users push the total arrival rate to 2000 titles per hour, which is compatible with BALLOTS planning activities, the prime shift CPU utilization will still be only 19.8% with total response time of 4.24 seconds.

Since these numbers are so low and also represent an upper bound on the BALLOTS central system interactive performance, other interactive processing models, for example the Network File Design, were not run.

4.3.2 BEST/1 BALLOTS Batch Model

By dividing title processing into an interactive and a batch component, BALLOTS is able to transfer the larger portion of the processing load to the non-prime shift. As can be seen from the BEST/1 interactive model, this leads to an extremely large effective capacity during prime shift. However, it does lead to a significant batch load which must be cleared each night.

A BEST/1 model of the BALLOTS batch file update processing was developed to determine performance as a function of load. BALLOTS batch was modeled as a single workload on the dedicated machine, locked into main memory (i.e., no swapping) and processing file requests which were randomly distributed over the six disk spindles.

Four different resource usages (current, tuned, Network File Design original and Network File Design copy) for BALLOTS batch update were identified and discussed at length above. The BEST/1 input parameter values to which they correspond are shown in Table 6.

Currently, BALLOTS batch is processed single thread, i.e., multiprogramming level of one. This will still be the case when the tuned batch update program goes into production. BALLOTS planning for the Network File Design is also oriented around single thread

TABLE 6

BALLOTS BATCH	CURRENT	TUNED	NFD	NFD-COPY
CPU time (in seconds)	.591	.473	.500	.150
Total # of I/O's*	62	45	42	14
DISK1 time (in seconds)	.341	.248	.231	.077
DISK2 time (in seconds)	.341	.248	.231	.077
⋮	⋮	⋮	⋮	⋮
DISK6 time (in seconds)	.341	.248	.231	.077

* Assumption that each 2K-byte I/O requires 33 msec of disk service time; also I/O's are randomly distributed over 6 disks.

processing since this design simplifies the software recovery logic. Stanford personnel have stated that when the Network File Design is operational, total elapsed time for update processing will not be a problem. However, on a contingency basis, they would consider restructuring batch update to be multi-thread in the unlikely event that it should become necessary.

The results of the BEST/1 batch model support Stanford's position. Table 7 presents the results of 24 different BEST/1 batch model runs. The throughput (in titles processed by batch per hour) and CPU utilization are shown for multiprogramming levels one through six, for each of current batch, tuned batch, Network File Design original cataloging, and Network File Design copy cataloging.

TABLE 7

BALLOTS BATCH	CPU					
	1	2	3	4	5	6
CURRENT	22.1	39.0	51.6	61.4	69.2	75.4
TUNED	24.1	41.8	55.1	65.4	73.3	79.6
NFD	26.5	45.7	59.9	70.6	78.6	84.7
NFD-COPY	24.5	42.4	55.9	66.3	74.3	80.5

BALLOTS BATCH	THROUGHPUT					
	1	2	3	4	5	6
CURRENT	1365	2373	3141	3740	4214	4594
TUNED	1835	3180	4194	4975	5582	6058
NFD	1909	3290	4314	5082	5661	6097
NFD-COPY	5882	10185	13423	15905	17827	19322

As can be seen, even with the single thread tuned batch, the 1978 client workload of 1600 titles per day (200 titles per hour times eight hours) can be processed in less than one hour while the 1983 client workload

10 Based on 8 hours per day, 5 days per week and 50 weeks per year.

of 3200 titles per day can be processed in less than two hours.

However, Stanford's plan is to move to single thread in the Network File Design in order to substantially reduce batch resource usage. Under these circumstances, if the client workload were 75% copy cataloging and 25% original, the 1978 client batch update would take less than 25 minutes while the 1983 client batch update would take less than 50 minutes.

If the total daily BALLOTS workload grew to 10,000 titles (compatible with Stanford planning) and only 75% represented copy cataloging, the entire batch update could still be cleared with single thread in about 2 1/2 hours. If Stanford went to multi-thread update, higher throughputs with shorter elapsed times are achievable. However, the multi-thread throughput values shown in Table 7 are estimates since multi-thread processing may necessitate some software redesign.

5. Conclusion

The overall performance analysis methodology described above was applied to the other bids received by the client. Utilizing the agreed upon generic cataloging activity, we were able to derive both the resources consumed (per title) and the response time characteristics under the prescribed (short-term and long-term) loads. This information provided important quantitative input to the client during the evaluation process.

The authors would like to thank the many individuals who participated in the Automated Bibliographic Retrieval System Study. Particular thanks are due to John R. Schroeder and Hanan S. Bell of Stanford University for their assistance in obtaining the data which is reported on in this paper.

References

1. BGS Systems, Inc., "System Design and Performance Study of Proposals for an Automated Bibliographic System", Lincoln, MA, March 1978.

2. Buzen, J.P. et al., "BEST/1 - Design of a tool for computer system capacity planning", Proc. AFIPS Conf. 47 (1978 NCC), 447-455.

3. BALLOTS Network Reference Manual, BALLOTS Center, Stanford, CA, December 1976.

4. ORVYL/370 The Stanford Timesharing System - Functional Description, Stanford Center for Information Processing, Stanford, CA, October 1977.

5. Design of SPIRES II, Volumes I & II, Stanford Center for Information Processing, Stanford, CA, July 1973.

CPEUG 78

PERFORMANCE IMPROVEMENT PART I:
QUANTITATIVE METHODS

PERFORMANCE IMPROVEMENT - PART I: QUANTITATIVE METHODS

Aridaman K. Jain
Bell Laboratories
Holmdel, NJ 07733

1. Introduction

Computer Performance Evaluation (CPE) is concerned with measurement, analysis and evaluation of computer systems. The CPE studies are based on one or more of the following approaches:

(i) Analytical
(ii) Discrete Simulation
(iii) Empirical.

The analytic approach consists of describing the functioning of a computer system or one of its components as a mathematical model. Much of the analytic modeling has concentrated on exact solutions for modest models which are gross simplifications. Queuing theory and other branches of probability are used in the analytic approach.

The discrete simulation approach is used in lieu of an analytical study when the computer system is too complicated for an analytical study. One paper [1] in this session is on the application of discrete event computer simulation.

The empirical approach consists of (i) design of a study, (ii) collection of data, (iii) analysis of data, and (iv) interpretation of the results. Most of the empirical studies in the literature concentrate on the collection of data with very little emphasis on the analysis of the data and interpretation of the results. Therefore, very limited benefit has been derived from these empirical studies. Unfortunately, the data collection phase in most empirical studies does not use the techniques of design of experiments and consequently the data collection is very ineffective. Similarly, the data analysis phase in most empirical studies does not go far enough and it could benefit from the use of proper statistical methods.

The CPE studies in the literature generally lack the formulation of an objective function (necessary for evaluation) as well as the proper use of quantitative methods. This session attempts to encourage CPE analysts to use quantitative methods for evaluation and improvement of computer performance. Four papers [2,3,4,5] in this session are concerned with the empirical approach. The use of quantitative techniques discussed in these papers will enhance the value of CPE studies.

2. Review of Past Empirical Studies

In order to examine the strengths and weaknesses of past empirical CPE studies, it is useful to review some of these. The review in this section is based on a tutorial [6] presented at the last meeting of the Computer Performance Evaluation Users Group (CPEUG). This review is divided into two parts: (i) studies deficient in the use of statistical techniques, and (ii) studies which made good use of statistical techniques.

2.1 Studies Deficient in the Use of Statistical Techniques

A majority of the empirical CPE studies emphasize the collection of data, with very little emphasis on their analysis and interpretation. A few papers presented at the 12th meeting of CPEUG in San Diego [7] are reviewed.

"A Performance Study of the Multiprocessor Transition in a Large Computer System" [7] was concerned with the effect of upgrading a single CPU system to a dual CPU system. The average throughputs for single CPU and dual CPU were compared without any consideration of the inherent variability in the process. The analysis of variance technique (discussed in Section 4.4) could have been applied to estimate the components of variability due to CPU types as well as the week-to-week variability under the same CPU type. This study ignored the effects of (i) additional disk units and (ii) growing workload.

It is claimed that these effects are small. But, these effects could have been easily estimated through fitting of regression models. It is better to first estimate effects and ignore them <u>only if</u> they are found to be small.

"Honeywell Time Sharing Experiments on a CPU Bound System" [7] reported on the effects of (i) increased workload and (ii) certain parameter changes. The results consisted of average response times without any considerations of inherent random variability. By comparing the average response times it is claimed that the lengthening of the time slice (given by TSS to its subsystems) improves the response. Had a proper statistical analysis been done by fitting a regression model, and asking "whether the fitted regression coefficient associated with the length of time slice might be merely an estimate of zero", the answer would probably have been "yes". Why? Even though the raw data are not available for such regression modeling, an examination of the following averages (read off a figure from the reviewed paper) supports the above inference regarding the regression coefficient associated with the length of time slice:

Experiment No.	Time Slice (Milliseconds)	Average Response Time (Seconds)
12	25	20.2
14	25	19.2
15	15	20.8
20	35	18.4

The experiments 12 and 14 are identical. The average response times for these two experiments differ only because of random variability. This random variability can account for the observed differences between the 25 milliseconds time slice experiments and the other two experiments with time slices of 15 and 35 milliseconds respectively, which would indicate that the length of time slice has no effect on response time. Thus, it is not sufficient to consider just the averages without their associated variabilities.

"A Simulation of TSO" [7] discussed a simulation system designed to estimate the effects of increasing use of TSO. This paper gave a detailed description of the process of data acquisition and pointed out an important fact: "Accurate identification, acquisition and organization of the data is the most critical and time consuming phase of a project". This paper presented the average responses for both the previous parameters and the recommended parameters which showed the advantage of the recommended parameters. It would have been useful to do the following: (i) to estimate the inherent variability, (ii) to fit smooth curves for each set of parameters, and (iii) to judge the improvement for the recommended set in light of the inherent variability. Here again, a more thorough statistical analysis may have changed some of the conclusions reached in this paper.

2.2 Studies Which Made Good Use of Statistical Techniques

There are several studies in the literature which have made good use of statistical techniques. Some of these studies are reviewed below.

Schneidewind [8] discusses a study of correlation between performance and resource usage variables and then development of regression models. He also used a linear programming model to select an optimum job mix. First, he analyzed the correlation matrices to identify important variables for regression. The multiple correlation coefficient, residual mean square and an examination of residuals were used in measuring the adequacy of the fitted regression models.

Schatzoff and Bryant [9] discuss two examples of regression in performance evaluation. In addition to developing the regression models, they discuss problems which may arise and appropriate corrective measures. For example, they point out the problem of correlated workload variables which makes it difficult to interpret any single regression coefficient. However, they could have carried out a better analysis of residuals by making plots and examining them.

Yeh [10] describes a method for identifying proper descriptor variables which may be used in fitting regression equations. Physical interpretation is provided for the fitted equations. The following possible applications of the fitted model are discussed: (i) calibration of workload and environments through regression equations, and (ii) study of abnormal or unusual system conditions which may be indicated by outliers.

Tsao and Margolin [11] present the statistical methodology applied in their analysis of a multi-factor paging experiment. They statistically designed an experiment to study the effect of four factors (main memory size, problem program, deck arrangement and replacement algorithm) upon the paging process. All 81 possible combinations (3 levels of each factor) of the four factors were observed. The preliminary step in the analysis of data was the computation and plotting of summary statistics. For the response variable, number of page swaps (PS), the range of 81 responses was about eight times their average. This prompted the rescaling to logarithms. Regression models were fitted and the technique of analysis of variance was employed to decompose the variability of the response into components attributable to various factors. This decomposition permitted an examination of the relative importance of the effects of the various factors in the modeling of the response.

The paper by Tsao and Margolin [11] is quite exhaustive in coverage of the statistical methodology for their multi-factor paging experiments. I recommend it for your study.

3. Papers in This Session

As indicated earlier, papers in this session are concerned with the use of quantitative methods for evaluation and improvement of computer performance. The topics covered in this session include the following: (i) multidimensional data analysis, (ii) time series, (iii) interval estimation, (iv) discrete simulation, and (v) resource control. Applicability of certain quantitative techniques to the problems of evaluation and improvement of computer performance is illustrated through examples.

Schroeder [2] describes some techniques of multidimensional data analysis, which are applicable to measurements of computer systems. Application of the techniques of clustering and geometric representation to the following problems is illustrated: (i) workload characterization, (ii) program addressing, (iii) program restructuring, and (iv) data base allocation. This paper is effective in indicating some practical problems encountered in the analysis of real data and how to cope with some of these problems.

Kulp and Melendez [3] describe an application of time series analysis to computer performance evaluation. The following time series models are discussed: (i) autoregressive, (ii) moving average, and (iii) autoregressive-moving average. Data from two computer systems are used to fit models and to forecast workload for future planning. A good feature of this paper is the examination of residuals for testing the adequacy of the fitted model.

Mamrak and Amer [4] discuss interval estimation of job run times as an alternative to point estimates. They use the signature table method, which is a technique for recognition of binary patterns. A good feature of this paper is the inclusion of a detailed example on a run time prediction problem for the U.S. Army.

Melendez and Linder [1] describe the analysis of a discrete event digital computer simulation of a computer. They characterize the computer workload in terms of five job parameters and three system parameters and develop a discrete event simulation model using SIMSCRIPT II.5. A highlight of this paper is the discussion of the sensitivity analysis of the response surface with respect to job I/O time and job CPU time.

Mackinder [5] discusses a statistical approach to resource control in a large time sharing system. A gamma probability distribution was found to be a good fit to computer usage. The proposed approach allows machine capacity to be managed as a whole and dispenses with the rationing system, which was in use earlier. A major advantage of this approach is its flexibility to cope with sudden changes in the number of computers jobs.

References

[1] Melendez, K., and Linder, A. H., "Sensitivity Analysis and the Response Surface of Simulation Model", Proceedings of the 14th CPEUG Meeting, 1978.

[2] Schroeder, A., "Multidimensional Data Analysis as a Tool for the Study of a Computer System", Proceedings of the 14th CPEUG Meeting, 1978.

[3] Kulp, R. W., and Melendez, K., "An Application of Time Series Analysis in Computer Performance Evaluation", Proceedings of the 14th CPEUG Meeting, 1978.

[4] Mamrak, S. A., and Amer, P. D., "Estimation of Run Times Using Signature Table Analysis", Proceedings of the 14th CPEUG Meeting, 1978.

[5] Mackinder, C. A., "A Statistical Approach to Resource Control in A Time-Sharing System", Proceedings of the 14th CPEUG Meeting, 1978.

[6] Jain, A. K., "Statistical Approaches in Computer Performance Evaluation Studies: A Tutorial", presented at the 13th CPEUG Meeting, 1977.

[7] Proceedings of the 12th CPEUG Meeting, 1976.

[8] Schneidewind, N. F., "Analysis of Computer Performance in Multiprogrammed Processing", 9th-11th CPEUG Meetings, 1974-75.

[9] Schatzoff, M., and Bryant, P., "Regression Methods in Performance Evaluation: Some Comments on the State of the Art", Proceedings of Computer Science and Statistics: 7th Annual Interface, 1973.

[10] Yeh, A. C., "An Application of Statistical Methodology in the Study of Computer System Performance", Proceedings of a Conference on Statistical Computer Performance Evaluation, edited by W. Freiberger, 1971.

[11] Tsao, R. F., and Margolin, B. H., "A Multi-Factor Paging Experiment: II. Statistical Methodology", Proceedings of a conference on Statistical Computer Performance Evaluation, edited by W. Freiberger, 1971.

HOW MULTIDIMENSIONAL DATA ANALYSIS TECHNIQUES CAN BE OF HELP IN THE STUDY OF COMPUTER SYSTEMS

Anne Schroeder

IRIA Laboria
BP 105
78150 Le Chesnay
France

Various measurements of computing systems have some common features: the phenomenon to be observed depends on several factors the influences of which cannot be considered individually and also the samples can be very large. The purpose of this paper is to give general ideas on Multidimensional Data Analysis methods which are able to process such samples. They are descriptive statistical techniques dealing with multidimensional samples which ensure their ability to take multiple factors into account. Some applications in the field of computer systems analysis illustrate the use that can be made of those methods: program behaviour, workload characterization, data base organization.

Key words: Clustering; Correspondence Analysis; data base organization; Multidimensional Data Analysis; Principal Components Analysis; Workload Characterization.

1. Introduction

Collecting measurements and processing them is a crucial step in the study of computer systems. In performance evaluation for instance, when techniques such as modeling and simulation are used, data are needed to calibrate and validate the models or the simulators. To compare allocation strategies in a data base management system, performance indicators have to be defined and measured. To manage a computer centre efficiently, it is important to have a precise knowledge of the workload and of its fluctuations. In such studies, once measurements are collected, their processing requires the use of statistical tools. Various measurements of computing systems have some common features. Generally, the phenomenon to be observed may depend on many factors the individual influences of which cannot be considered separately. Also, the samples can be very large; the behaviour of a program, for instance, can be described by a reference string of several millions observations.

In this paper, we describe Multidimensional Data Analysis (MDA) techniques which are able to process samples presenting the above features. They are descriptive statistical techniques dealing with multidimensional samples; this ensures their ability to take multiple factors into account. Though the basic principles of these tools have been known for some time, their recent development is based on the availability of powerful computation means. The techniques require numerical algorithms for matrix diagonalisation and inversion which are executable only by computers as soon as the dimensions of the matrices are larger than two or three.

In section 2, we present the multidimensional point of view with the help of some examples taken from the field of computer systems analysis. Section 3 provides a technical introduction to a wide family of multidimensional data analytic methods. Applications of these methods to the problems stated in section 2 are presented in section 4. Those applications indicate some of the practical problems that arise when analysing real data. Section 5 presents some concluding remarks.

2. The Multidimensional Aspect of Some Problems Arising in Computer Systems Analysis

We give here some examples of practical problems for which the multidimensional point of view has been proven to be useful.

2.1 Example 1 - Workload Characterization

As pointed out by D. Ferrari [Fer72], the problem of describing precisely and of characterizing the workload of a system (or of a whole computer centre) is an important step in the fields of both management and performance evaluation. It consists of understanding better what kinds of resource needs correspond to different parts of the workload. This understanding is necessary to approach several practical problems: for instance, building benchmarks that are representative of a given workload; finding discriminant parameters to schedule programs according to their requirements; improving both system efficiency and user's satisfaction, and so on.

Measurements of workloads are generally performed as follows: during some representative period of activity, for each job (or step) information is collected about its nature and its various requirements in time, space, peripherals, etc... Thus are built as many statistical samples as observed variables. Classical works on workload characterization consists of the isolated study of those samples; at most, regression analysis has been used to examine the influence of a set of parameters on one privileged variable. On an other hand, the multidimensional approach permits consideration of all possible interactions between parameters by characterizing each step by a set of variables such as: i^{th} step = (arrival time, CPU time used, core required, core used, number of disk I/O, number of tapes mounted).

This sequence consists of heterogeneous information which can be coded as some vector of real numbers. We shall discuss the coding problem specifically in § 4. The workload can then be considered as a set of vectors in some space of high dimensionality. Multidimensional Data Analysis is then able to identify clusters of steps (or step types) with similar resource needs. Such analyses are presented in section 4.

2.2 Example 2 - Program Addressing Behaviour

The presence of "localities" [Den68] is an interesting feature of the memory referencing behaviour of most programs. It means that the execution time may be divided into distinct phases during which the program references are concentrated in some favoured set of addresses. The concept of locality of memory references is of great importance to the design of virtual memory systems, and it has been used explicitly in various memory management policies implemented in such systems. Despite its intuitive simplicity, the concept of locality is hard to quantify because of its relativeness: for a same program, there may be different locality structures when different levels of detail are considered. Several a priori models have been proposed [ShT72, BaS76, FrG75]; but if one wants to approach locality a posteriori as a physical observation of a program behaviour, two questions can be asked: (1) Having observed executions of a program, how many and which localities does it run through? (2) Knowing the localities liable to be encountered by a given program, identify which one is being crossed at a given instant during the execution?

In a previous paper [Sch77b], we proposed a tool to determine the localities that also provided an easy method to identify them afterwards. It was based on the multidimensional description of the "working-set string" of the program. Based on the notion of "working set" [Den68], it is the sequence of the characteristic functions of the successive working-sets sampled with some given sampling period. Each sampled working-set is represented in the following way: i^{th} working-set: $(x_{i1}, x_{i2},\ldots,x_{iN})$ where N is the number of pages in the program virtual space and x_{ij} equals 1 if the j^{th} page is referenced in the i^{th} working-set and zero otherwise. Thus, the dynamic behaviour of a program can be represented as a sequence of vectors in $\{0, 1\}^N$. Detecting localities or stationary regimes during the execution of a program can be stated as detecting classes of short time intervals during which the program continues to use the same portions of its address space. This can be done by clustering those similar working-set vectors.

2.3 Example 3 - Program Restructuring

The purpose of program restructuring techniques is to improve the efficiency of paging mechanisms by an adequate display of the program in a virtual memory. In order to do that, programs are first cut up into several blocks; these blocks are determined by the programmer, and therefore they are generally logical modules composing the program. Thus, the problem is to map those blocks into the virtual pages in order to optimize a performance index (for example, to minimize the number of page faults occuring during the execution). This implies that blocks that are frequently accessed one after the other are assigned the same pages. Many restructuring methods have been presented in the literature [Fer76, MaS74, HaG71, AcB77]. They have in common the fact that they deal with a similarity (or dissimilarity) matrix between the blocks. That is, each block is characterized by the vector of its similarity indices to all the others and the information needed for restructuring is contained in that whole vector. The restructuring problem can then be stated in terms of clustering the blocks with respect to their similarity indices. This is possible with the help of multidimensional techniques.

2.4 Example 4 - Data Base Allocation or Reorganization

In large data bases implemented on storage hierarchies, it is important to distribute records into groups so that each transaction requires retrieval of as few of those groups as possible. In order to do that, clusters of records that are frequently referenced by the same transactions have to be identified for being clustered together. To know how user's transactions reference records, one may collect those records retrieved by each transaction during some representative activity period. A record can then be represented by the following vector: i^{th} record = $(x_{i1}, x_{i2}, \ldots, x_{iM})$ where x_{ij} equals 1 if the j^{th} transaction uses the i^{th} record and zero otherwise, with M being the total number of sampled transactions. This representation follows the same pattern as that introduced in the Example 2: the addressing units are records on one hand and pages on the other, while the elementary activity units are transactions on one hand and working-sets on the other. In the same way, the problem of data base reorganization can be stated in terms of clustering record vectors.

On the other hand, it can also be interesting to study the behaviour of the different transactions with respect to the records they use. For this purpose, one can represent the j^{th} transactions as a vector $(x_{1j}, x_{2j}, \ldots, x_{Nj})$ if N is the total number of records and x_{ij} are the same as above. Clusters of transactions can be used to restructure commonly used programs on the data base.

These few examples, chosen from the field of computer systems analysis, show how the multidimensional approach can be used in various applications. The following section will discuss multidimensional data analysis techniques which are applicable to such examples.

3. Descriptive Methods in Multidimensional Data Analysis (MDA)

The examples presented above have shown how several practical problems can be addressed by their representation as a finite set of high dimension vectors. Let us denote by R^p the Euclidean p-dimensional space, that is the set of all p-vectors of real numbers. Let n be the number of vectors (or points) of R^p in the sample. If p equals 1 or 2, that is if one or two variables are observed, many well known classical techniques can give a description of the sample: histograms, goodness-of-fit tests with some presumed distribution, correlation computation and significance tests, regression analysis of one variable on the other. In regression analysis, the sample is represented as points in R^2, with each coordinate axis representing one of the two variables; then the general structure of the sample is visible on a plane graph: clusters may be observed or a linear dependence between the two variables suggested. When the number of variables is greater than two, such a direct indication of relationship is much more difficult. The general purpose of the methods we shall present is to provide a synthetic representation of multidimensional sets.

Section 3.1 introduces notation as well as one of the basic concepts in MDA: similarity (or dissimilarity) between vectors. Then the different methods are presented according to the way they represent a multidimensional data set: first, description by geometrical plane representations (or by projection); the description by clustering, and eventually description by graphs.

3.1 Distance and Similarity Measures

From now on, the basic information we shall use is the data table (traditionally called an "observation-parameter" table), denoted by X.

$$X = \begin{array}{ccccc|c} & & x_{1j} & & & 1 \\ & & \vdots & & & \vdots \\ x_{i1} & \cdots & x_{ij} & \cdots & x_{ip} & i \\ & & \vdots & & & \vdots \\ & & x_{nj} & & & n \\ \hline 1 & \cdots & j & \cdots & p & \end{array}$$

where each element x_{ij} is a real number and x_{ij} is the value of the ith observation of the jth parameter.

Let us denote

column j by : $\underline{v}_j = \begin{pmatrix} x_{1j} \\ \vdots \\ x_{ij} \\ \vdots \\ x_{nj} \end{pmatrix}$

and row i by: $\underline{x}_i = \begin{pmatrix} x_{i1} \\ \vdots \\ x_{ij} \\ \vdots \\ x_{ip} \end{pmatrix}$

So, $\underline{v}_j$ is the vector in R^n representing the n-sample of the jth parameter, and $\underline{x}_i$ is the vector in R^p representing the p values of the parameters measured at the ith observation.

The set we want to describe is $E = \{\underline{x}_i / i = 1,\ldots,n\}$. We may immediately note that "observations" and "parameters" formally play symmetrical parts. Any method dealing with a n-sample of points in R^p may of course be applied to a p-sample of points in R^n. The methods we shall present will therefore be able to describe the set $E = \{\underline{x}_i \;/\; i = 1, \ldots , n\}$ of the observations as well as that of the parameters, $F = \{\underline{v}_j \;/\; j = 1, \ldots , p\}$.

In order to analyze a multidimensional data set, the first thing needed, besides the data table itself, is a measure of similarity (or dissimilarity) between observations; any method will have to represent the observations according to their mutual resemblances. It is of primary importance to determine exactly what we mean by "these two observations are close to one another" or by "this observation looks more like this one than like this other one." This requirement may be fulfilled by the choice of a distance d or a similarity measure on the space of observations (here, R^p).

Let us recall that a distance function d on the set E takes as its arguments two vectors of E, say $\underline{x}_i$ and $\underline{x}_\ell$, and produces a non-negative real number $d(\underline{x}_i, \underline{x}_\ell)$ such that:

i) $d(\underline{x}_i, \underline{x}_\ell) = 0 \Longrightarrow \underline{x}_i = \underline{x}_\ell$

ii) $\forall i, \forall \ell, d(\underline{x}_i, \underline{x}_\ell) = d(\underline{x}_\ell, x_i)$

iii) $\forall i, \forall \ell, \forall m, d(\underline{x}_i, \underline{x}_\ell) \leq d(\underline{x}_i, \underline{x}_m) + d(\underline{x}_m, \underline{x}_\ell)$

A function d which only satisfies (i) and (ii) will be called a "dissimilarity measure." Hereafter a dissimilarity measure has to be chosen or given, and this choice is fundamental. Results obtained by the same technique and from the same data may be completely different for different definitions of "closeness."

3.2 Description by Geometrical Representations

These methods consist in choosing in the p-dimensional space of the data a subspace of low dimension (generally 2, 3, etc...) such that the projection of the data set on this subspace keeps as much information as possible about the original p-dimensional set. There are as many projection methods as notions of "good" representation. In the following, we shall present a classical one: Principal Components Analysis (PCA) and one of its fruitful extensions: Correspondence Analysis (CA). These two methods assume that a projection is good if the projected points are as close as possible to the original ones in terms of the chosen distance measure on R^p. Thus the projection subspaces will be found by minimizing dispersion. Other notions of a good projection can also be considered: The Projection Pursuit Algorithm [FrT74] satisfies some trade-off between a good representation of the global dispersion and that of local clusters; Discriminant Analysis ([And58], for instance) gives a representation in which given clusters of the sample are as well separated as possible. Let us also note the various techniques of Proximity Analysis and Multidimensional Scaling [She62, Kru64] which attempt to give a low-dimensional representation such that the distance table of the represented set is as close as possible to the original one.

3.2.1 Principal Components Analysis (PCA)

Let X be the data table, and $E = \{\underline{x}_i / i = 1,\dots,n\}$ the set of n points in R^p to be analyzed. The space R^p is assumed to have a Euclidean distance function d_M, that is a distance that verifies Pythagoras' theorem. Let us assume that weights p_i are assigned to the n observations. (If there is no prior information on the sample, these p_i may be taken all equal. However, it may happen that natural weights arise. This is the case for instance when there are several identical observations of the variables. They can then be represented by only one of them the weight of which is proportional to their number).

Then, the problem of Principal Component Analysis (PCA) is to find a subspace of R^p of low dimension such that the projection of E on this subspace is a "good" representation of E. Generally, we shall look for 2, 3, 4 and 5-dimensional subspaces and observe them in their various plane cross sections.

More precisely, the problem of PCA can be stated in the following way:

$$\left[\begin{array}{l} \text{Find a subspace } E_k \in R^p \ (\dim(E_k) = k << p) \text{ such that} \\ \sum_{i=1,n} p_i d_M^{\,2}(\underline{x}_i, E_k) \text{ is minimum.} \end{array}\right.$$

(Let us recall that the distance between a point and a subspace is the distance between that point and its M-orthogonal projection on the subspace).

Details on PCA techniques and on the validity of the low dimension representations obtained may be found in [And58]. Once the set E of observations have been represented, the set F of parameters may also be represented by performing another PCA on the data table X', the transposition of X. Generally, there is no analytical relationship between the two representations so they must be interpreted independently. In the next section we present a particular version of PCA which allows simultaneous interpretation of the two representations.

3.2.2 Correspondence Analysis (CA)

Due to J. P. Benzecri [Ben73], Correspondence Analysis (CA) is a particular PCA. We have seen that, besides the data table, PCA requires two more inputs from the user: a weighting system $\{p_i / i = 1,\dots,n\}$ and a distance, associated with a matrix M, on R^p. In CA, those two inputs are chosen so that the two PCA dealing respectively with the two sets E and F of observations and of parameters, may be easily deduced from one another and simultaneously interpreted.

For this purpose, the chosen distance is a classical statistical distance, called χ^2-distance, on E as well as on F [And58, Ben73]. The weights in each set are the marginal frequencies obtained by summing rows and columns in the data table. With those choices, the two representations of E and F are related by simple analytical relations. This relationship has two practical consequences:

- the description of the two sets E and F requires only one anlaysis. The other one can then be easily deduced from the first.

- the two representations can be interpreted simultaneously. The structure of the set of observations can thus be explained in terms of the parameters and vice-versa.

The benefits of using and interpreting Correspondence Analysis will be illustrated in the examples of Section 4.

3.3 Description of Clustering

Up to now, the methods we have seen describe multidimensional data sets by means of geometrical representations, illustrating visually some properties of the observed populations. In this paragraph, we shall present methods that partition the data set. To make precise the notion of a "good" partition, it is necessary to introduce a criterion function on all possible partitions of the set E of observations. The different clustering methods differ from one another either by the quality criterion satisfied by their solution, or by the procedure leading to a solution satisfying a given criterion.

Given the data set E in R^p and some distance d on R^p, the most frequently used criteria are based on the dispersions of the classes of the partition. For example, a common criterion of quality for the partition $P = (P_1, \dots, P_k)$ in k classes of E (i.e.; $E = \coprod_{1 \leq \ell \leq k} P_\ell$ and the P_ℓ are disjoint) is the

3.4 Description by Graphs

What we called "geometrical" representation in § 3.2 were graphical representations which were to be read and interpreted as maps: both individual and global notions of closeness were reasonable. In this section, we present representations by graphs where the notions of nodes, edges and paths are used to interpret the structure of the data.

All the methods in this section work on distance tables D, (i.e., the set E to be anlayzed is characterized by all distances between pairs of its elements). Whether these distances are the actual data or they have been obtained from some representation or using some particular distance does not matter.

Thus these methods do not use directly the multidimensional representation of the set to analyze as a set of points in a p-dimensional space. It is the distance structure of this set which is explicitly taken into account. The multidimensional point of view underlies the distance based methods and, in most applications, both approaches are taken jointly.

3.4.1 Hierarchical Clustering

Hierarchical clustering is a representation of the data by a hierarchical tree (or dendrogram). It also provides homogeneous nested partitions of the data; this is why it is called "hierarchical clustering." At the lower level of the tree all the points to be analyzed are represented. Then the tree is built by aggregating those points or previously formed clusters that are the closest to each other. Each level of the tree represents a partition of the set E. An extensive study of hierarchical clustering methods is made in [SnS73].

3.4.2 Graph Partitioning Methods

The knowledge of the distance table D permits one to assign a graph structure to the set E of objects to analyze. It can be considered as a complete graph with edge lengths given by the distances, possibly weighted as well. Then E is the set of nodes and for all i and ℓ in E, there is an edge (i,ℓ) the length of which is $d_{i\ell}$. If E is very large, there are methods for building an incomplete graph on E such that the existing edges bring as much information as possible on the complete distance graph. Two such methods are:

- The construction of the Minimum Spanning Tree (MST) of E [Zah71]: it is a spanning tree (i.e. connex and having all the points of E as nodes), and among all spanning trees, its length is minimum. It has the following property: the distance $d_{i\ell}$ between any points i and ℓ of E is at least as great as the longest edge in the unique path joining i and ℓ in the MST.

- Methods for reducing the sizes of both sets of nodes and edges of the initial complete graph on E [MiD77]. The number of nodes is reduced by aggregating those points in E that are strongly related in the initial graph; the number of edges is then reduced by techniques similar to that of the MST.

In any case, once the set of E is represented as a graph, graph partitioning methods can be applied to partition E. In [Ker71], Kernighan proposes an algorithm to find a minimum cost partition of the nodes into given size classes. This partition is optimum for a given order of the nodes. In [GoR69], a clustering method based on the MST is given: by partitioning the set of the nodes of the MST by suppressing its longest edge and by iterating this process inside each class obtained, one gets a sequence of nested partitions of E. (This sequence is proved to be the same as that given by hierarchical clustering when all the distances in D are different [Ben73]).

4. Applications of MDA to Practical Examples

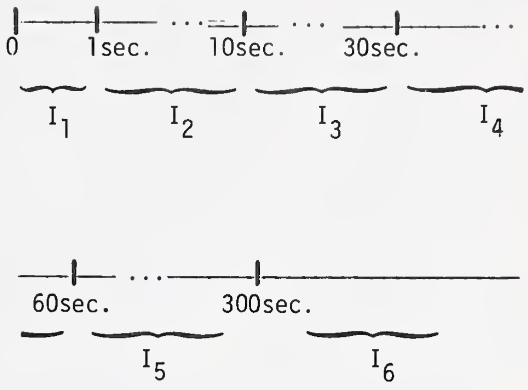
In the previous section we have presented some techniques in multidimensional data analysis. Of course, their ulilization has to be preceeded by a thorough anlaysis of the problem in order to be able to describe it precisely, to choose a sampling strategy and to select important parameters. The tools presented take a data table X as a starting point and the work involved in acquiring such a data table for some practical context should not be underestimated. Some of the problems that can arise during this preliminary work are mentioned in the presentation of applications which follows.

4.1 Workload Characterization

A workload can be described in terms of clusters of worksteps characterized by several variables. Agrawala et al. set up the problem in that way [AgM76]; the worksteps are characterized by eight parameters (CPU time, number of I/O on different devices, number of steps executed, executive charges, etc...); they are clustered by using a variant of the "k-means" algorithm (cf. § 3.3.iii) and eventually the clusters are interpreted with the help of their Kiviat graphs [KoK73] on the eight variables. In a complementary paper [AgM77], Agrawala and Mohr give further results on the influence of the set of variables selected on the clustering obtained. Different sets of variables lead to completely different clusters; this could be expected since it is different multidimensional sets which are clustered: this emphasizes the importance of the choice of variables. This difficulty can be partially alleviated by "feature selection" procedures that help determine a relevant set of parameters to describe a workload [MaA77]. In such procedures, known jobs are initially classified into distinct groups. The feature selection methods then give those parameters which are sufficient to describe adequately those groups. A similar approach in the characterization is presented by H. P. Artis [Art76]. He gives a description of a job step by a "resource vector," thus characterizing the steps of the workload by their physical resources requirements. The steps are then clustered with the help of the ISODATA algorithm (cf. § 3.3.iii).

Workload analyses using MDA techniques have been recently conducted in France. Since a few of these studies have been published [Bio78, Car77, Duc78, Kem76] we shall indicate briefly their common features and the kind of results they yield. Different workloads are studied: either the whole load of a given computer or its batch processing load or the load of a whole computer center. Also, different elementary activity units are considered as "observations" in our statistical vocabulary: either steps, or jobs, or customers. In any case, the parameters measured are the requirements for the same main resources: CPU, core memory, different types of disks, tapes, etc.. All authors come to a data table as that described in § 2.1 the columns of which are heterogeneous variables (e.g. a processing time, a number of disk accesses per minute, a number of cards punched,...).

To transform this kind of mixed information into a data table manageable by MDA techniques, the approach which has been adopted in the above given references is to transform each continuous variable into several binary variables by subdividing its range into significant classes (such classes must be as equiprobable as possible and must also take into account any characteristic feature of the histogram like isolated peaks). In [Bio78] for instance, the CPU time range is divided into the following six intervals:



Then the continuous variable "CPU time" is represented by six binary variables: I_1, I_2, I_3, I_4, I_5, I_6. Each observation takes the values "1" for one and only one of those six variables and the value "0" for the others. For instance, to one step which required 1.6 seconds of CPU time, there will correspond the six-tuple of CPU time variables: (0, 1, 0, 0, 0, 0). Once such a coding has been performed for all the parameters, the data table is a large binary table that can be analyzed by some of the MDA techniques presented above. In the three studies mentioned above, Correspondence Analysis (cf. § 3.2.2) has been applied and though the definitions of the parameters differ from one another, the main results are the same. As an example, we give on figure 1 the representation of the parameters obtained by J. Biondi [Bio78].

Along the first principal axis are ordered the binary variables representing CPU time and main core requirements in increasing order. This shows the preponderous importance of those two parameters as well as the fact that they are related to each other. At the same time, the representation of the steps (not given here) gives a general scaling of the steps (or jobs) from "small" (short and with small core memory requirement) to "big" (long and requiring much core space). This seems to be a general feature in the workloads analyzed in those studies. The remaining principal axes point out various influences as that of fast or slow disk I/O, that of tapes, and so on. A careful interpretation of these graphs gives a picture of the load and of its characteristic features. To go further and acquire a precise classification of the steps (or jobs, or customers) according to their uses and requirements, clustering methods can be applied. In [Bio78] and [Car77], iterative following:

$$W(P) = \sum_{1 \leq \ell \leq k} \sum_{\underline{x}_i \in P_\ell} d^2(\underline{x}_i, \underline{g}_\ell)$$

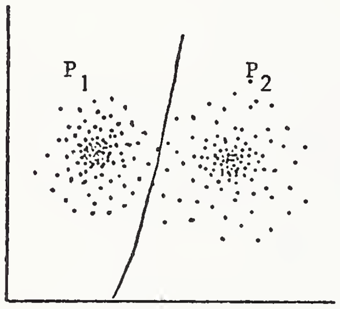
where $\underline{g}_\ell$ is the centre of gravity of the ℓth class P_ℓ of the partition, i.e.:

$$\underline{g}_\ell = \frac{1}{\text{Card } (P_\ell)} \sum_{\underline{x}_i \in P_\ell} \underline{x}_i$$

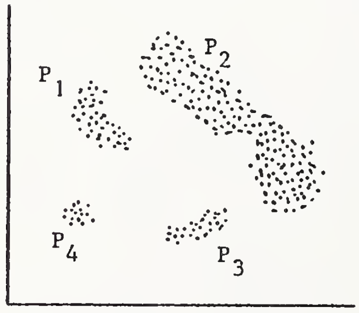
The criterion W(P) is to be minimized, resulting in a situation in which points which are "close" to each other are in the same class of the partition.

Different combinations of these dispersions, with various weights for instance, may also be used as criteria for good partitions. Other criteria may be defined that take into account the presence of "gaps" between groups of points, or the shapes of these groups, for example.

Let us give two examples where "natural" partitions seem to arise while they behave quite differently in terms of dispersions:



Example 1



Example 2

Example 1 shows a partition into 2 classes (P_1, P_2) which is a "good" partition for the dispersion criterion defined above. The example 2 shows a "natural" partition into 4 classes though the criterion it optimizes would be quite difficult to define.

Detailed studies of various criteria and of clustering techniques can be found in [Rub67, FrR67, Mar76, Dor71]. The first method that may come to mind for finding the partition P of E into k classes which optimizes a given criterion function W is to compute W(P) for all P, and choose the best one. Of course, this method is inordinately expensive. Other methods avoid this complete enumeration by introducing some *a priori* information or by yielding solutions which are only locally optimal. Clustering techniques can be roughly classified into three families:

i) "Threshold" methods: these methods generally take into account the data sequentially and build clusters progressively while introducing them. Thresholds are used to limit either the maximum (or minimum) volume of the clusters or their ("nearest-neighbour" procedures). We will not go into further details on those methods since we are not aware of any application in the area of computer systems. For more details, see [Bon64, CoH67, Dor71].

ii) Manipulations on the distance (or similarity) matrix: given the similarity matrix between all the items to be clustered, a "good" clustering should be such that the reordered similarity matrix (after the clustering) is as close as possible to a semi-block diagonal form. Starting from this simple observation, McCormick *et al* [McS72] have developed the "Bond Energy Algorithm." This algorithm is an efficient method for permuting the rows and columns of the similarity matrix until a satisfactory diagonal form is obtained.

iii) Iterative methods: these are methods in which initial classes are drawn at random and iteratively improved. This improvement is achieved by aggregating the clusters around some characteristic "centroid." The techniques differ from each other by the notion of centroid they use: it may be the centre of gravity (ISODATA [BaH67]), or some subset of the class (Dynamic Clusters [Did72, DiS74]; k-means [McQ67]; for a complete review, see [Cor71, Dor71]). All these methods lead to local optima of the criterion.

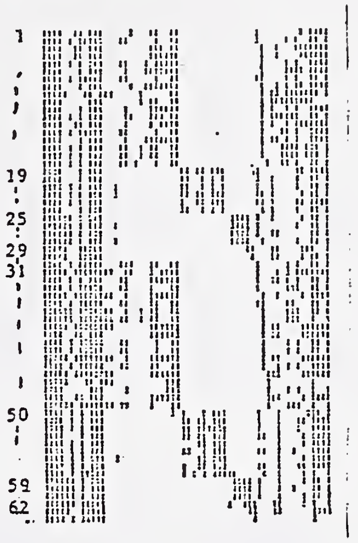
clustering techniques such as those described in § 3.3.iii have been used to get classes of characteristic jobs, while hierarchical clustering (cf. § 3.4.1) has been applied to the set of the variables in [Bio78]. In practice, hierarchical methods cannot be used for sets as large as those of the jobs, while iterative methods can deal with such sets. It is also possible to pass from a clustering of the variables (smaller set) to a clustering of the steps (which can be a very large set) with the help of the transition formulae of Correspondence Analysis (cf. § 3.2.2) [Bio78]. In [Bio78], the classification obtained and its characteristic parameters have been used to construct synthetic jobs that could be used as benchmarks. In [Car77], the results have been used to study the evolution of the workload and similar analyses have been carried out and compared month after month. Customer profiles have also been defined.

4.2 Program Addressing Behaviour

Detailed analyses of working-sets tables such as those presented in § 2.2 have been thoroughly described in [Sch77b]. We shall briefly review one of them here in order to illustrate the practice of MDA. Let us recall that a "working-set" [Den68] is the set of pages referenced by a program during a given virtual time interval. The working-set strings we consider are obtained by sampling working-sets on intervals of fixed length, expressed as a number of references and known as the 'window size." The working-set string considered here is that of a FORTRAN compiler. It has been extracted at the University of Rennes on a 10070 CII Computer [BuL76]. The window size is 5000 references and the interval between two successive sampled working-sets is 50,000 references; the corresponding data table is given on figure 2.

On this data table, we first carried out a Correspondence Analysis (see § 3.2.2). Figures 3 and 4 show the representations of the set of the working-sets in the first two principal planes (axes 1 and 2 and axes 1 and 3). The parts of information carried by the three first axes are: 21%, 15%, 10%.

On the first principal plane (Fig. 3), four groups obviously appear which correspond to the four phases that are encountered two times by the program and that were distinguishable on the initial string of figure 2. When observing the second plane (Fig. 4), one notes that the three phases 2, 3, 4 are still clustered together while the phase 1 is split into two subphases, one of which is very close to the phase 4. These subphases can also be distinguished in the initial string. Eventually the representation of the working-sets in a 3-dimensional space indicates clearly the existence of four different phases through which the program runs twice. Figure 5 shows the representation of the pages on the first principal plane. This is the symmetrical analysis of the previous one.



1
19
25
29
31
50
59
62

Figure 2

Each line represents a working-set, each column a page, the working-sets are numbered from top to bottom in chronological order (the "0's" of the table have replaced by blanks).

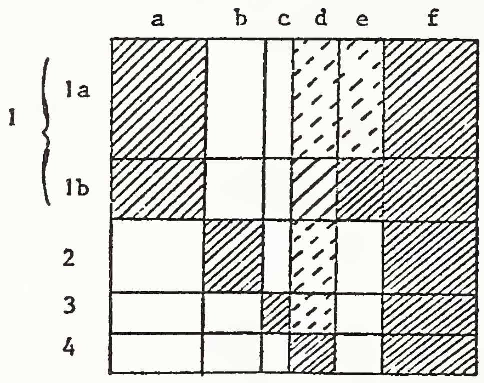
When observing figure 5 and figure 3 simultaneously, one can make the following observations. On figure 5, the most frequently referenced pages are gathered. They are accessed throughout the execution and do not characterize any phase. The others are clustered according to a pattern similar to the one displayed on figure 3. One can then observe which are the groups of pages which characterize the above determined phases and which are those that are intermediate between two of them. For instance, pages 33, 34, 37, 39, 40, 41, 42 are characteristic of phase 2; pages 45, 46, 47, 48 of phase 3 and pages 6, 49 of phase 4 while pages 50, 17, 35, 44 may as well appear in any of the three phases 2, 3, 4. From the representation of the pages on their second principal plane (not given here), it is also possible to see which page references differ in the two subphases 1a and 1b.

After the Correspondence Analysis has been interpreted, a simultaneous clustering on the two sets related by the data table [Gov77] has been performed. The clustering of the set of working-sets is the same as the one deduced from Correspondence Analysis:

five classes 1a, 1b, 2, 3, 4. The clustering of the pages gives the following six classes:

Class a:	18, 19, 20, 21, 22, 23, 24, 26, 29, 30, 64, 65, 67, 69
Class b:	33, 34, 35, 36, 37, 39, 40, 41, 42
Class c:	45, 46, 47, 48
Class d:	6, 10, 44, 49, 62, 63, 70
Class e:	8, 14, 15, 27, 28, 52, 71
Class f:	2, 3, 4, 7, 9, 11, 12, 13, 17, 50, 51, 72, 73, 74, 75

This clustering is included in the one deduced from Correspondence Analysis and is consistent with it. The mutual relationships between the two sets are summarized in the following table:



This table is deduced from the initial data table by regrouping lines and columns according to the clustering obtained and with:

▩	more than 80% of "1" in the cell
▨	from 40 to 60% of "1"
▧	from 10 to 30% of "1"
□	empty cell

Percentages between 60 and 80%, 30 and 40%, 0 and 10% have not been observed.

The interpretation of this table is immediate; for instance, phase 1 is characterized by the use of the pages belonging to classes a, d and e; a is very frequently used during the whole phase and though d and e are less frequently referenced, it is their use which discriminates between the two subphases 1a and 1b.

The multidimensional aspect of a string of working-sets which has been indicated above gives a very convenient tool for characterizing localities. We have seen how the sampled working-sets may be considered as vectors, how they can be compared to one another by using a distance measure (here the χ^2 distance), and how a locality can be defined in terms of simultaneous clustering of those vectors and of the set of pages. Thus a new sampled working-set may be identified as belonging to one of the detected localities by measuring its distance to the different clusters of working-sets and assigning it to the closest one.

Another use of the multidimensional aspect of working-set strings is mentioned by Freiberger et al. [FrG75]. They present a model of program behaviour based on the notion of "regimes." Their assumption is that a program runs through different independent stationary regimes which are characterized by their vectors of page reference frequencies. In order to partition regimes in page reference strings they identify the current regime on some time interval by calculating the page reference frequencies on this interval; then, passing to a following interval, it will be said to be in the same regime or in another one according to whether the distance (usual Euclidean distance) between the two page frequencies vectors in small or large. This example shows the utility of the geometrical and multidimensional points of view in studying such problems.

4.3 Program Restructuring

As presented in § 2.3, the problem of restructuring programs can be stated in terms of clustering the blocks of the program. This clustering has two main characteristics:

- it is based on a distance (or a similarity) measure between the blocks,
- it has to take into account constraints of size for the clusters. This is because the program has to be displayed in pages of the virtual memory which are of some fixed size.

Authors dealing with the restructuring have chosen various measures of proximity between the blocks. In [HaG71], the similarity between two blocks is exactly the number of times one of the blocks has referenced the other one during one execution (number of transfers). In [MaS74], two blocks are closer the more often they have been referenced together in some given time interval of the execution of the program.

In [AcB77], the similarity between two blocks is also based on the number of transfers from one to the other, but it is somehow refined by dividing this number by the total number of transfers; this "relative" number of transfers seems to give a more precise idea of how close two blocks are. D. Ferrari [Fer76] has proposed several similarity measures, which are based on the notion of "critical" references: a critical reference is a reference which may give rise to a page fault depending on the layout chosen for the program. The problem is thus stated in terms of clustering blocks related to one another by a distance (or similarity) matrix. Moreover, this clustering must satisfy a size constraint (the page size).

The clustering techniques that we have presented in § 3.3 and § 3.4 (hierarchical and non-hierarchical clustering) do not take any constraint into account; so, they have to be modified to solve restructuring problems. Since there does not exist any optimal clustering method under size constraints, the restructuring algorithms make use of heuristic variants of known clustering algorithms. In [AcB77] for instance, the hierarchical clustering (cf. § 3.4.1) is used, but the respect of the size constraint is to be ensured at each step of the aggregation. The same clustering method is used in [MaS74] but the size constraint is only taken into account a posteriori at the end of the algorithm. Among the different variants of clustering algorithms he proposes, D. Ferrari suggests a simple stepwise aggregation method with a size threshold; this technique is close to the clustering threshold methods mentioned in § 3.3.i.

4.4 Data Base Reorganization

To improve data base reorganization as stated in § 2.4, S. Gorenstein and G. Galati [GoG74] propose to cluster the records of the base on the basis of the data table "record x transactions" defined in § 2.4. The results they present have been obtained by using a modified descending hierarchical clustering algorithm (cf. § 3.4.1). The criterion used for splitting is the dispersion of the clusters. The improvement in the performance of the system after the clustering of the records seems significant and encouraging. In the same paper, the authors show how a variant of the "k-means" methods (cf. § 3.3.iii) could also be used and give better results at a reasonable price.

In [HoS75], J.A. Hoffer and D.G. Severance set the problem of determining "an efficient segmentation for a data base which is shared by a community of users." Such a segmentation is an organization of the data base into subfiles containing only classes of records and classes of attributes. Thus, their purpose is to find those classes of records and attributes thay may constitute efficient subfiles. In [HoS75], they present the step of the clustering of attributes. In order to do that, they define an attribute similarity measure. The similarity between two attributes is based on the frequency of co-accesses to the two attributes in a set of sampled requests. Various weightings allow consideration of the user's priority and the value encoding length of each attribute. Eventually, once the attribute similarity table is built, the Bond Energy Algorithm (see § 3.3.ii and [McS72]) is applied to it, yielding groups of similar attributes. Then performance measurements have been made on several control runs in which the attributes were assigned to subfiles according to the results of the clustering. The results show the limits of the efficiency of the reorganization which are mainly related to the number of groups being considered. Efficiency is also sensitive to the choice of the weighting parameters in the attribute similarity definition.

On a lower level of data base problems, the same kind of techniques can be applied to the physical allocation of data to disks. In [FlG78], for instance, a variant of the Dynamic Clusters method (see § 3.3.iii) is used to improve the response time of a moving-head-disk. In a given file, those records that are frequently used in the same requests are clustered together on the basis of a sample of observed requests. Then, each cluster is assigned to a given cylinder.

5. Concluding Remarks

The purpose of this paper has been to present some descriptive multidimensional statistical tools. Emphasis has been put on applications in the area of computer systems analysis. Those applications cover a range of problems which are quite wide and the advantages of using Multidimensional Data Analysis Techniques.

Research is still open in directions of both methods and applications. Its success depends on the development of relationships between statisticians and computer system evaluators. In the use of MDA, like in any statistical application, the part played by the practicioner is substantial. He has to be involved at each step of the analysis; first, to state his practical problem in terms of descriptions of some finite set of data items; then to characterize these data items by relevant and manageable variables and to define the meaning of "closeness" or "similarity" between data items; finally, to choose some MDA technique and possibly adapt it to the problem. One could have noted in the applications presented that, most often, it is a "variant" of a known method which is used. Actually, there are no universal methods and each application raises specific problems which must be solved in its particular context. Eventually, the practitioner has to interpret the results obtained and can give the statistician hints for further work. We hope that this paper will encourage collaboration between statisticians and computer scientists.

References

[AcB77] Achard, M.S., Babonneau, J.Y. and Morisset, G., Automatic and General Solution to the Adaptation of Programs in the Paging Environment, Sixth ACM Symposium on Operating Systems Principles, Purdue University, November 1977 (or Research Report IRIA-Laboria No. 196).

[AgM76] Agrawala, A.K., Mohr, J.M. and Bryant, R.M., An Approach to the Workload Characterization Problem, Computer, June 1976, pp. 18-32.

[AgM77] Agrawala, A.K. and Mohr, J.M., Some Results on the Clustering Approach to Workload Modelling, Technical Report TR-521, Computer Science Technical Report Series, University of Maryland, April 1977 (or in the Proceedings of CPEUG 77).

[And58] Anderson, T.W., Introduction to Multivariate Statistical Analysis, Wiley, New York, 1958.

[Art76] Artis, H.P., A Technique for Determining the Capacity of a Computer System, Proceedings of CPEUG 76, November 1976, pp. 150-162.

[BaH67] Ball, G.H. and Hall, D.J., A Clustering Technique for Summarizing Multivariate Data, Behavioral Sciences, Vol. 12, No. 2, 1967, pp. 153-155.

[BaS76] Baer, J.L. and Sager, G.R., Dynamic Improvement of Locality in Virtual Memory Systems, IEEE Trans. on Software Engineering, Vol. SE-2, No. 1, March 1976, pp. 54-62.

[Ben69] Benzecri, J.P., Statistical Analysis as a Tool to Make Patterns Emerge from Data, in Methodologies of Pattern Recognition, Watanabe, S. (Ed.), Academic Press, New York, 1969, pp. 35-60.

[Ben73] Benzecri, J.P., et al., L'Analyse des Données, Vol. 2, Dunod, Paris, 1973.

[Bio78] Biondi, J., Description de la Charge d'un Système Informatique et Application à l'Evaluation des Performances, to appear in Revue Francaise d'Automatique Informatique Recherche Opérationnelle (Paris), Série B, Informatique, 1978.

[Bon64] Bonner, R.E., On Some Clustering Techniques, IBM Journal of Research and Development, Vol. 8, No. 1, 1964, pp. 22-32.

[BuL76] Burgevin, P. and Leroudier, J., Characteristics and Models of Program Behavior, National Conference of the ACM 76, Houston, October 1976, pp. 344-350.

[Car77] Carreiro, S., Utilisation de l'Analyse des Données Pour la Gestion d'un Centre de Calcul Etude de la Charge de l'Ordinateur, Thèse 3ème Cycle, Laboratoire de Statistique Mathématique, Université Paris VI, France, October 1977.

[CoH76] Cover, T.M. and Hart, P.E., Nearest-Neighbor Pattern Classification, IEEE Trans. on Information Theory, Vol. IT-13, No. 1, 1967, pp. 21-27.

[Cor71] Cormack, R.M., A Review of Classification, J. Roy. Statistical Society, Series A. Vol. 134, Part 3, 1971, pp. 321-353.

[Den68] Denning, P., The Working-Set Model for Program Behavior, Communications of the ACM, Vol. 11, May 1968, pp. 323-333.

[Did72] Diday, E., An Introduction to the Dynamic Clusters Method, METRA, Vol. XI, No. 3, 1972, pp. 505-519.

[DiS74] Diday, E., Schroeder, A. and Ok, Y., The Dynamic Clusters Method in Pattern Recognition, Information Processing 74 (Proc. IFIP Congress 1974), North-Holland Publishing Company, 1974, pp. 691-697.

[Dor71] Dorofeyuk, A.A., Automatic Classification Algorithms (Review), Automation and Remote Control, Vol. 32, No. 12, Part 1, December 1971, pp. 1928-1958.

[Duc78] Duclos, A., Mesures des Performances et Analyse de la Charge d'un Calculateur Multiprogrammé, Thèse de Docteur-Ingénieur, Conservatoire National des Arts et Metiers, Paris, June 1978.

[Fer72] Ferrari, D., Workload Characterization and Selection in Computer Performance Measurement, Computer, July/August 1972, pp. 18-24.

[Fer76] Ferrari, D., The Improvement of Program Behavior, Computer, November 1976, pp. 39-47.

[FlG78] Flory, A., Gunther, J. and Kouloumdjian, J., Data Base Reorganization by Clustering Methods, Information Sciences, Vol. 3, Pergamon Press, 1978, pp. 59-62.

[FrG75] Freiberger, W.F., Grenander, U., Sampson, P.D., Patterns in Program References, IBM Journal of Research and Development, May 1975, pp. 230-243.

[FrR67] Friedman, H.P. and Rubin, J., On Some Invariant Criteria for Grouping Data, Journal of American Statistical Assn., Vol. 62, December 1967, pp. 1159-1178.

[FrT74] Friedman, J.H. and Tukey, J.W., A Projection Pursuit Algorithm for Exploratory Data Analysis, IEEE Trans. on Computers, Vol. C-23, No. 9, September 1974, pp. 881-890.

[GoG74] Gorenstein, S. and Galati, G., Data Base Reorganization for a Storage Hierarchy, IBM Research Report No. RC5063, Yorktown Heights, October 1974.

[GoR69] Gower, J.C. and Ross, G.J.S., Minimum Spanning Trees and Single Linkage Cluster Analysis, Applied Statistics, Vol. 18, No. 1, 1969, pp. 54-64.

[Gov77] Govaert, G., Algorithme de Classification d'un Tableau de Contingence, in Analyse de Données et Informatique, Colloque IRIA, Rocquencourt, France, 1977, pp. 487-500.

[HaG71] Hatfield, D.J. and Gerald, J., Program Restructuring in Virtual Memory, IBM Systems Journal, Vol. 10, No. 3, 1971, pp. 168-192.

[Har72] Hartignan, J.A., Direct Clustering of a Data Matrix, Journal of American Statistical Assn., Vol. 67, 1972, pp. 123-129.

[Har75] Hartignan, J.A., Clustering Algorithms, J. Wiley, New York, 1975.

[Hil74] Hill, M.O., Correspondence Analysis: A Neglected Multivariate Method, Applied Statistics, Vol. 23, No. 3, 1974, pp. 340-354.

[HoS75] Hoffer, J.A. and Severance, D.G., The Use of Cluster Analysis in Physical Data Base Design, Proceedings of the Very Large Data Base Conference, D.S. Kerr (Ed.), 1975, pp. 69-86.

[Kem76] Kempf, J.B., Application de l'Analyse Multidimensionnelle à des Travaux Soumis au Calculateur de Clamart, in La Statistique: Outil d'Analyse des Systèmes Informatiques, D. Potier and A. Schroeder (Eds.), French Chapter of the ACM, 1976.

[Ker71] Kernighan, B.W., Optimal Sequential Partitions of Graphs, Journal of the ACM, Vol. 18, No. 1, January 1971, pp. 34-40.

[KoK73] Kolence, K.W. and Kiviat, P.J., Software Unit Profiles and Kiviat Figures, Performance Evaluation Review, Vol. 2, No. 3, September 1973, pp. 2-12.

[Kru64] Kruskal, J.B., Non Metric Multidimensional Scaling: A Numerical Method, Psychometrika, Vol. 29, 1964.

[LaW65] Lance, G.N. and Williams, W.T., Computer Programs for Monothetic Classification--Association Analysis, Computer Journal, 1965, pp. 246-249.

[LeS75] Leroudier, J. and Schroeder, A., A Statistical Approach to the Estimation of Service Times Distributions for Operating Systems Modelling, International Computing Symposium 75, E. Gelenbe and D. Potier (Eds.), 1975, pp. 171-184.

[MaA77] Mamrak, S.A. and Amer, P.D., A Feature Selection Tool for Workload Characterization - International Conference on Computer Performance Modelling, Measurement and Management, SIGMETRICS/CMG VIII, November 1977.

[MaB76] Madison, A.W. and Batson, A.P., Characteristics of Program Localities, Communications of the ACM, Vol. 19, No. 1, May 1974, pp. 285-294.

[MaS74] Masuda, T., Shiota, H., Ngughi, K. and Ohki, T., Optimization of Program Organization by Cluster Analysis, in Information Processing 74 (Proceedings of IFIP Congress 74), North-Holland Publishing Company, 1974, pp. 261-265.

[McQ67] McQueen, J., Some Methods for Classification and Analysis of Multivariate Observations, Fifth Berkeley Symposium on Mathematics, Statistics and Probability, Vol. 1, No. 1, 1967, pp. 281-297.

[McS72] McCormick, W.T., Schweitzer, P.J. and White, T.W., Problem Decomposition and Data Reorganization by a Clustering Technique, Operations Research, Vol. 20, No. 5, September 1972, pp. 993-1009.

[MiD77] Milgram, M., Dubuisson, B. and Vachon, B., A Computationally Efficient Clustering Algorithm, IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-7, No. 2, February 1977, pp. 99-104.

[Rub67] Rubin, J., Optimal Classification Into Groups: An Approach for Solving the Taxonomy Problem, J. Theoret. Biol., Vol. 15, 1967, pp. 103-144.

[Sch76] Schroeder, A., Analyse d'un Mélange de Distributions de Probabilité de Meme Type, Revue de Statistique Appliquée, (Paris), Vol. XXIV, No. 1, 1976, pp. 39-62.

[Sch77a] Schroeder, A., Estimating Input for Operating Systems Models, in Recent Developments in Statistics, Barra et al. (Eds.), North-Holland Publishing Company, 1977, pp. 721-728.

[Sch77b] Schroeder, A., A Statistical Approach to the Study of Program Behavior via Reference Strings Analysis, in Computer Performance, Chandy and Reiser (Eds.), North-Holland Publishing Company, 1977, pp. 381-396 (or Research Report IRIA-Laboria No. 240).

[She62] Shepard, R.N., The Analysis of Proximities, Psychometrika, Vol. 27, 1962.

[SnS73] Sneath, P.H.A., Sokal, R.R., Numerical Taxonomy, W.H. Freeman and Company, 1973.

[SoS63] Sokal, R.R. and Sneath, P.H.A., Principles of Numerical Taxonomy, Addison Wesley, 1963.

[Zah71] Zahn, C.T., Graph Theoretical Methods for Determining and Describing Gestalt Clusters, IEEE Trans. on Computers, Vol. C-20, No. 1, 1971, pp. 68-86.

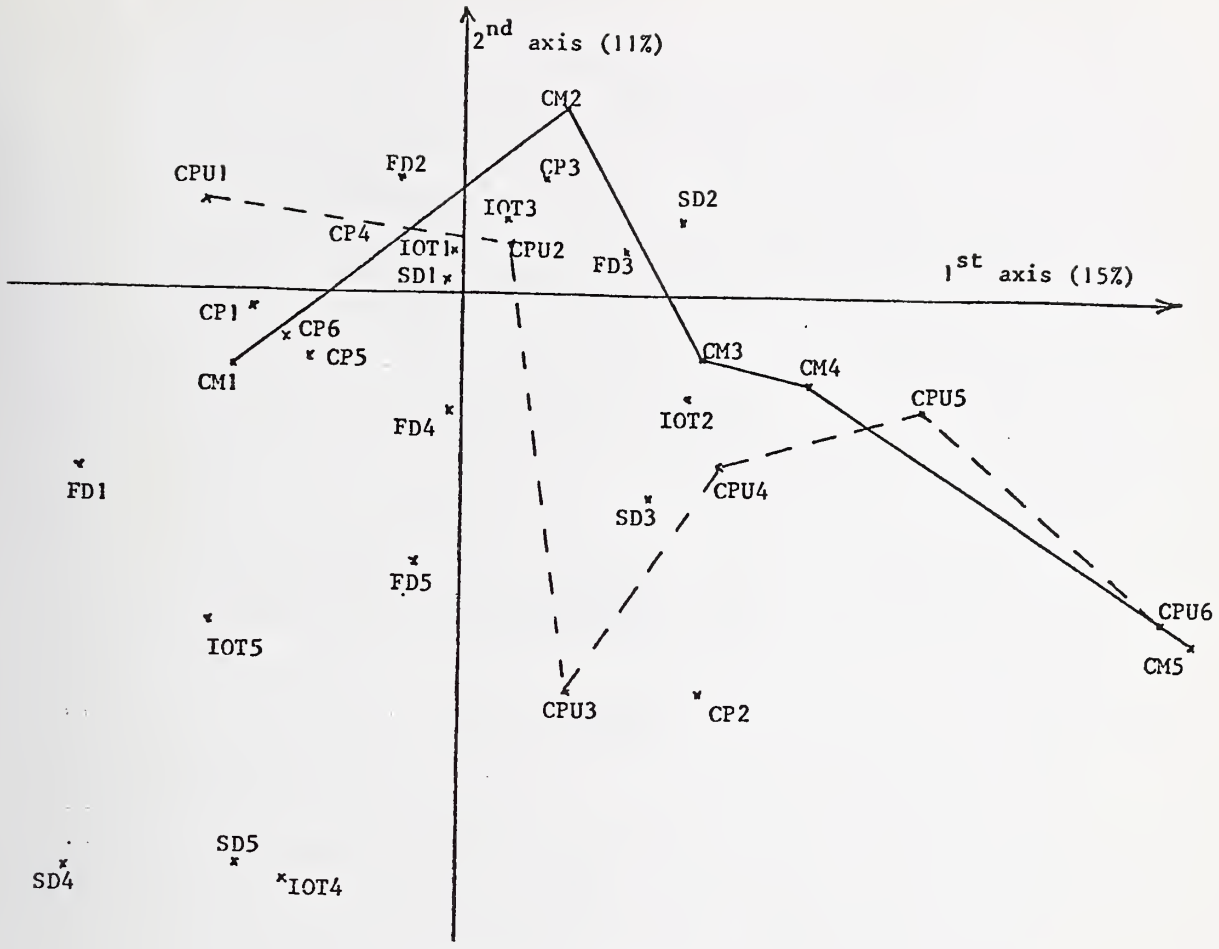


Figure 1 (from [Bio78])

Description of the variables:

- CPU time: variables CPU1 to CPU6 corresponding to $t < 1$ s, $1 \leq t < 10$ s, 10 s $\leq t < 30$ s, $30 \leq t < 60$ s, $60 \leq t < 300$ s, $t \geq 300$ s.
- Core memory used: variables CM1 to CM5 corresponding to $c \leq 68$ k, $60 < c \leq 136$ k, $136 < c \leq 256$ K, $256 < c \leq 400$ K and $c > 400$ K.
- Tape I/Ø: variables IØT1 to IØT5 representing the number of I/Ø per second of CPU time, cut up into the following intervals: N=0, $0 < N \leq 10$, $0 < N \leq 100$, $100 < N \leq 100$, $N > 1000$.
- Disk I/Ø OL IBM disks 2316: variables SD1 to SD5, same definition as for the tape I/Ø.
- Disk I/Ø on IBM disk 3336: variables FD1 to FD5, id.
- Cards and printer I/Ø: variables CP1 to CP6, where the range of the number N of I/Ø per second of CPU time is cut up into the following intervals: N=0, $0 < N \leq 1$, $1 < N \leq 10$, $10 < N \leq 100$, $100 < N \leq 500$, $N > 500$.

The percentages on each axis represent the part of information they carry.

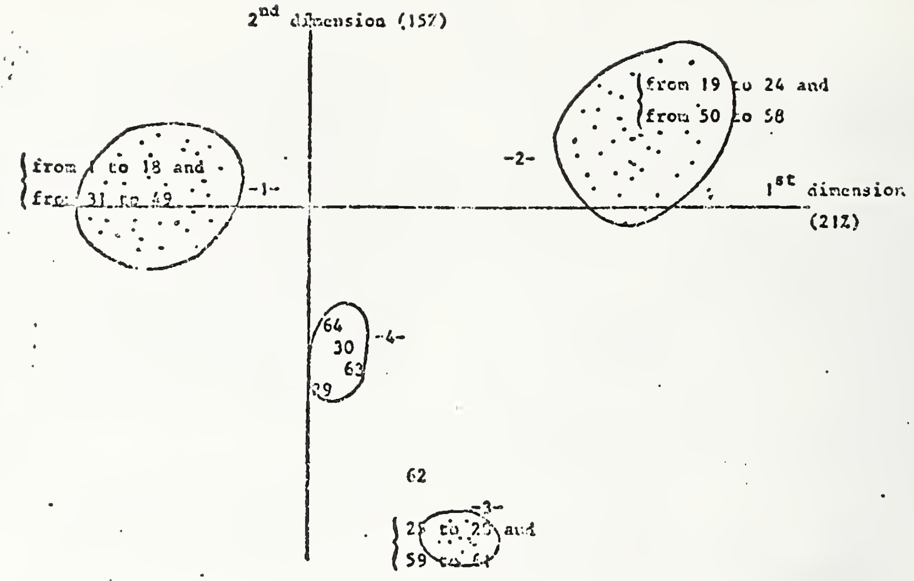


Figure 3

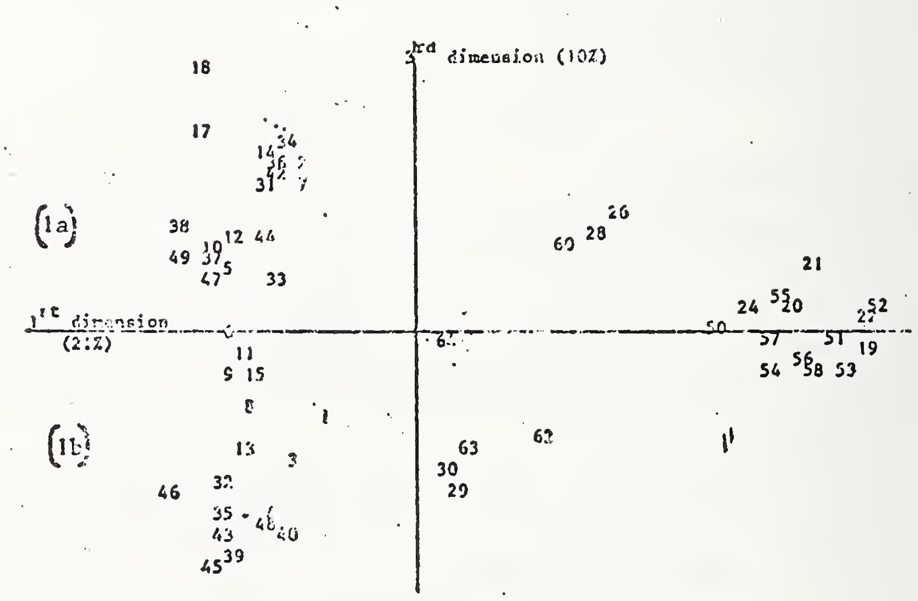


Figure 4

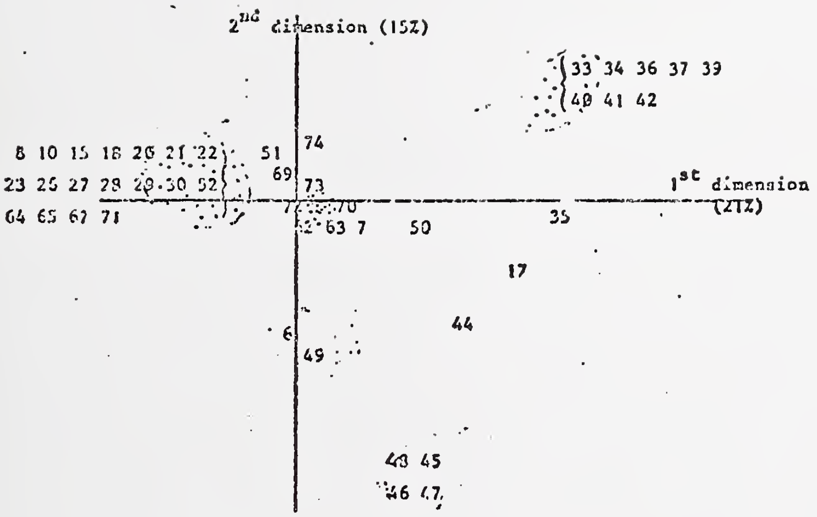

2nd dimension (15%)
1st dimension (21%)
8 10 15 18 20 21 22
23 25 27 28 29 30 52
64 65 67 71
51
69
74
73
70
2 63 7
50
35
33 34 36 37 39
40 41 42
17
44
6
49
43 45
46 47

Figure 5

AN APPLICATION OF TIME SERIES ANALYSIS IN COMPUTER PERFORMANCE EVALUATION

Major Richard W. Kulp
Major Kenneth Melendez

Air Force Institute of Technology
Wright-Patterson Air Force Base, Ohio 45433

One of the major roles of computer performance evaluation (CPE) in the life cycle is to predict the performance of a computer system. During any stage of the operating phase of a computer system it is possible to model the computer system as a time series and use that model to predict future performance. As an example, accurate prediction of when the computer system will transition to an overutilization or a saturation state will enable management to begin the life cycle of a follow-on system well before reaching system saturation on the current system. In this paper we show how the Box-Jenkins method of time series model building can be applied to measures of computer workload to provide predictions of future utilization. This technique is applicable to many measures of computer performance/workload. For this paper we have selected Computer Resource Units as a measure of performance. Actual data from a computer system that operates two computers, a CDC 6600 and a Cyber 74, has been analyzed and modeled using the Box-Jenkins method. The time series models developed are described and we demonstrate the use of these models in prediction.

1. Introduction

The singularly most important aspect of a computer system is its value to its user. Value to the user can be measured by the performance of the system and dollar cost per unit of time. By performance we mean how well a system performs the assigned tasks. We are assuming that the system performs correctly and we shall not discuss dollar cost but will focus our effort on a discussion of performance.

There are several methods of measuring the performance of a computer system such as throughput, central processor time used per day, and others. However, all of these measures of work have a common attribute. For each measure of performance, the observations of this measure are not independent from each other but, rather, form a sequence of dependent observations called a time series. Whereas many statistical procedures require independent observations, time series analysis procedures turn this dependence from a liability into an asset to capture the essence of the underlying process.

The method we propose to apply to Computer Performance Evaluation (CPE) problems is that advanced by Box and Jenkins [2][1]. There are at least two reasons for using this procedure:

(1) It provides a systematic method for identifying the type of model, estimating the parameters of the model and then performing diagnostic checks to determine the adequacy of the model.

(2) There are several computer packages available which calculate the required statistics.

[1]Figures in brackets indicate the literature references at the end of this paper.

Section 2 gives a mathematical formulation of the types of models which might prove useful in predicting workload (or some other factor, if desired) and gives some possible applications. It is not the purpose of this paper to derive or prove any mathematical results but to show that these models are "plausible" (see [1] and [2] for proofs). In Section 3 we analyze data from an existing computer system to demonstrate the method. A brief summary is given in Section 4.

2. Mathematical Formulation of Models

The models which we consider in this section assume that the underlying process is stationary in the sense that the joint probability distribution of the k observation $(X_t, X_{t+1}, \ldots, X_{t+k})$ depends only on the value of k and not the value of t. An immediate consequence of this requirement is that the covariance between observations X_t and X_{t+j} is a function only of the time difference, j, between X_t and X_{t+j} not the actual time values t and t+j. In order for this to be true there can be no trends (such as growth) in the data that is used for the model. We will see that this is usually not a serious problem as several techniques are available to "de-trend" the data. After "de-trending" the data is analyzed and a model fitted. Then the trend is put back into the model in order to carry out the prediction. The most common method of "de-trending" the observations is by "differencing", which is discussed later.

2.1. The Autoregressive Model (AR(p))

The autoregressive model of order p (AR(p)) describes the current observation of a time series as a weighted sum of p previous observations of the series plus a random perturbation which is independent of all previous observations. Mathematically, this is expressed as

$$X_t = \sum_{j=1}^{p} \beta_j X_{t-j} + Z_t, \qquad (2.1)$$

where $\{Z_t\}$ is a sequence of independent, identically distributed (i.i.d.) random variables with zero mean and variance $\sigma^2 < \infty$.

2.2. The Moving Average Model (MA(q))

Although aesthetically not very satisfying, it is often easier to model the current observation of a time series as a current perturbation plus a weighted sum of previous perturbations rather than a weighted sum of previous observations and a current perturbation. Such a model is called a moving average of order q where q is the number of previous perturbations in the weighted sum. Mathematically, this can be expressed as

$$X_t = Z_t + \sum_{k=1}^{q} \alpha_k Z_{t-k}, \qquad (2.2)$$

where $\{Z_t\}$ is a sequence of i.i.d. random variables with zero mean and variance $\sigma^2 < \infty$.

2.3. The Autoregressive-Moving Average Model (ARMA(p,q))

Often to adequately model a stationary time series one needs to combine both the AR(p) and MA(q) models mentioned above. In the resulting time series, the current observation is a weighted sum of both previous observations and perturbations plus a current shock. (At this juncture one might well exclaim, "Why bother?". The saving grace to all this is that one can usually get a "good" fit of the data with relatively small values of p and/or q, say 0, 1, or 2.)

Mathematically, the ARMA(p,q) model is written

$$X_t = \sum_{j=1}^{p} \beta_j X_{t-j} + \sum_{k=0}^{q} \alpha_k Z_{t-k} \qquad (2.3)$$

where the sequence of Z's are again i.i.d. with zero-mean, common variance $\sigma^2 < \infty$, and $\alpha_0 = 1$.

2.4. Detrending

All of the models above assume that the time series is stationary so that it behaves as random fluctuations about a common mean value. Obviously, this may not be realistic, especially if the workload is growing towards saturation and/or the activity of the system shows periodic behavior. Although there are many types of nonstationary behavior, the two types we will consider, polynomial or linear growth/decay trend or periodic trend, can often be taken out by differencing the observed series (possibly several times). For example, suppose the observed sequence $\{X_t\}$ displayed a general linear growth when plotted in time as in Fig. 1. By forming the differenced series $\{Y_t = X_t - X_{t-1}\}$ of length one less than $\{X_t\}$ we transform the series into one like that in Fig. 2. When differencing is necessary, the model is called the Autoregressive-Integrated-Moving Average Model (ARIMA(p,d,q)). Here p is the order of the autoregression, d is the number of times the series is differenced, and q is the order of the moving average part. Note that the AR(p) is exactly an ARIMA(p,0,0) and the MA(q) is just the ARIMA(0,0,q).

2.5. The Seasonal Model (ARIMA$(P,D,Q)\times(p,d,q)_s$)

Often we find that the data will follow a seasonal pattern. This could easily arise if the computer center has a production cycle--say weekly, monthly or quarterly. When this happens it is natural to expect that observations may be dependent not only on the immediate past but also on observations taken s units of time ago where s is the length of the cycle. For example, the daily workload of a computer center that has a monthly production cycle may depend on the amount of work for the past day or two and also the amount of work on the same day for the previous month or two.

In order to analyze this type of behavior it is necessary to remove the seasonal component and then analyze the resulting sequence. In removing trends or nonstationary behavior in nonseasonal time series we "differenced" the series X_t by forming the series $Y_t = X_{t+1} - X_t$, t = 1,2,3,...,n-1 to get a new series of length n-1. In a similar manner we remove the seasonal trend by differencing by the length of the period, i.e., if the data is monthly and there is a quarterly cycle then we difference by 3 and form the series $Y_t = X_{t+3} - X_t$, t = 1,2,...,n-3 of length n-3. The resulting series is denoted by $Y_t = \nabla_s X_t$ where s is the period. This series is then analyzed in terms of shocks s units apart in a manner similar to the ARIMA(p,d,q) models above except that now the shocks W_t may not be independent (usually they are not independent). Thus we have

$$\nabla_s X_t = \sum_{i=1}^{P} A_j \nabla_s X_{t-i} + W_t + \sum_{j=1}^{Q} B_j W_{t-j} \quad (2.4)$$

which is denoted by ARIMA$(P,D,Q)_s$. To adjust for the dependence of the W_t we write the W_t sequence as an ARIMA(p,d,q) model. When we do this we get what is called a multiplicative seasonal model ARIMA$(P,D,Q)\times(p,d,q)_s$.

2.6. Model Identification

Since a time series is a sequence of dependent observations one might expect that the correlation between observations in the sequence would play an important role. It turns out that in a stationary time series it is the autocovariance and autocorrelation functions which describe the behavior of the series and lead us to the identification of plausible models. Before proceeding, we need to define autocovariance and autocorrelation.

If X and Y are two random variables then recall that cov(X,Y) = E((X-E(X))(Y-E(Y)). If we were to substitute X_t for X and X_{t+h} for Y in the above definition we have the definition of the covariance of the sequence X_t with itself for times separated by h units; i.e., the autocovariance of X at "lag h", denoted by $\gamma(h)$. Thus

$$\gamma(h) = E((X_t-E(x_t))(X_{t+h}-E(X_t)). \quad (2.5)$$

(Since $\{X_t\}$ is stationary, the actual value of t is not important but rather the distance between successive observations, or lag, is important. Moreover, stationarity implies that $E(X_{t+h}) = E(X_t) = \mu_x$ for otherwise there would be some form of growth or decay). Using the $\gamma(h)$ notation we see that the variance of X_t is just

$$\text{Var}(X_t) = E(X_t-\mu_x)^2 = \gamma(0). \quad (2.6)$$

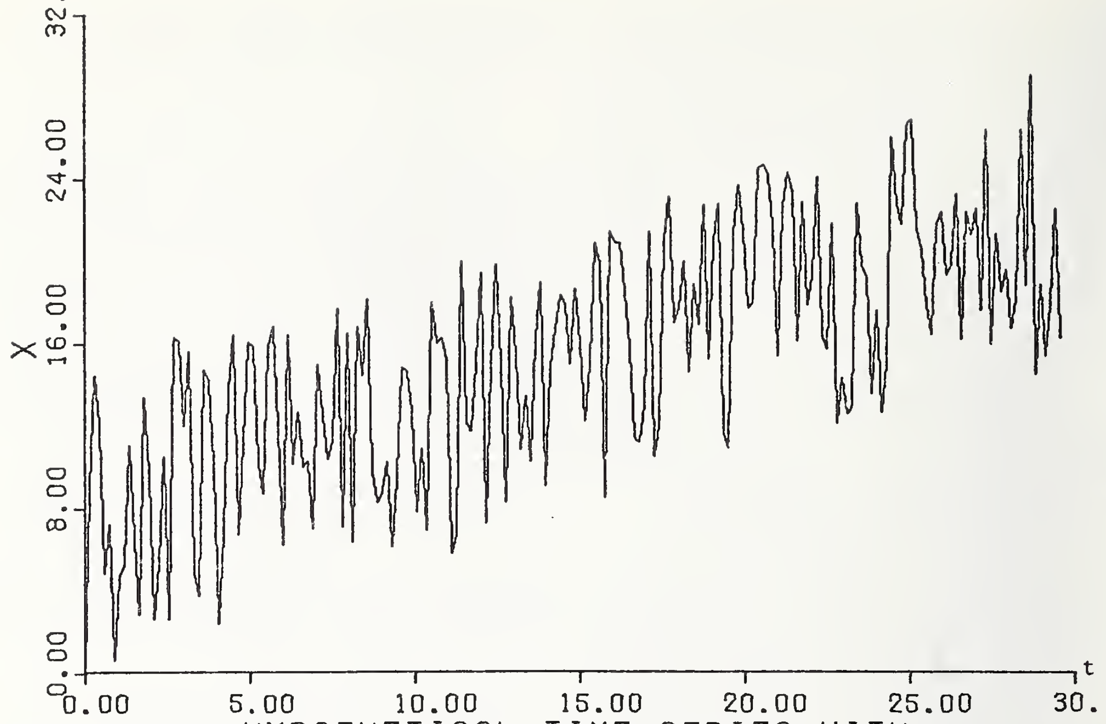
In order to place everything on the same footing, the autocovariance function is usually divided by the variance of X, $\gamma(0)$, to get the autocorrelation function, ρ_k. The autocorrelation function behaves as any correlation function in that $|\rho_k| \leq 1$ with $\rho_0 = 1$.

2.7. Estimation of $\gamma(k)$ and ρ_k

The least squares estimators of $\gamma(k)$ and ρ_k when the residual sequence $\{Z_t\}$ has finite variance is given by

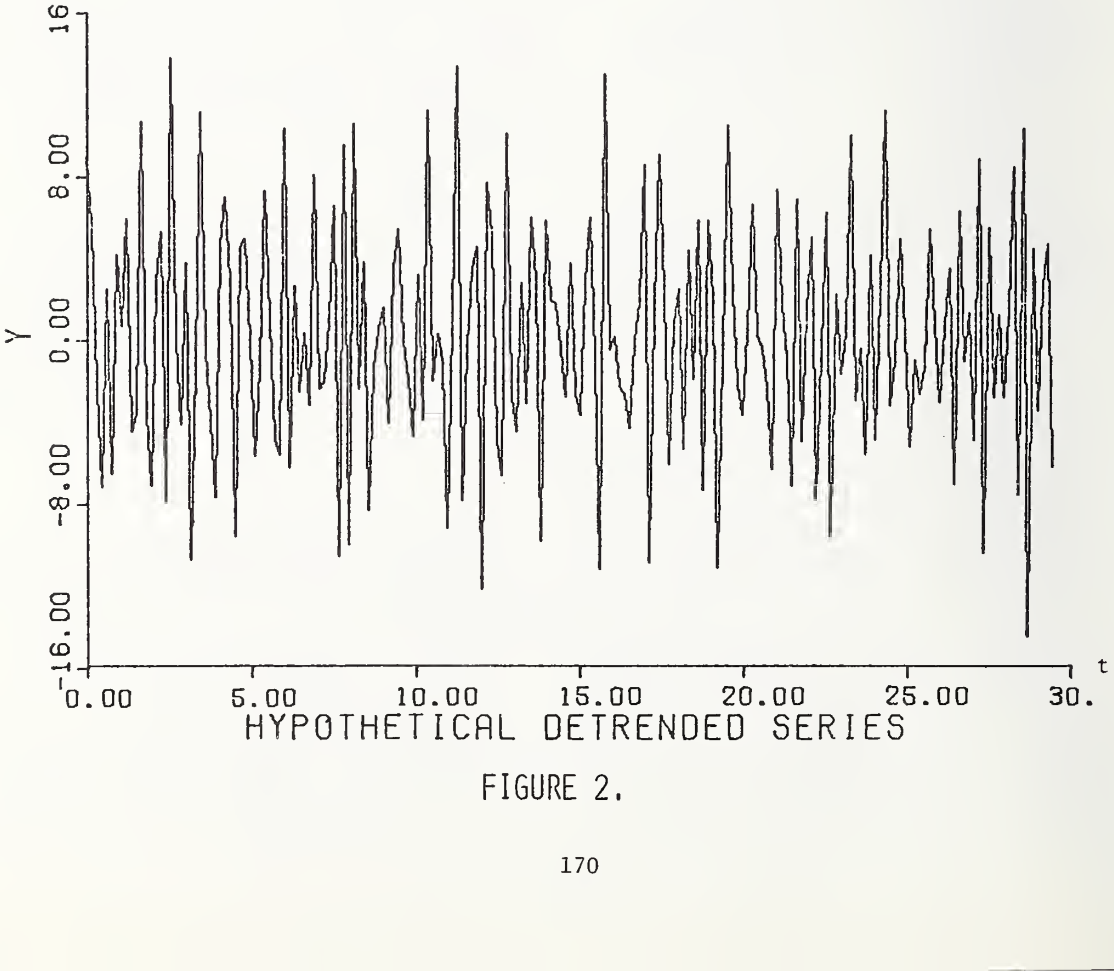
$$c_k = \frac{1}{N} \sum_{t=1}^{N-k} (X_t-\overline{X})(X_{t+k}-\overline{X}) \quad (2.7)$$

$$c_0 = \frac{1}{N} \sum_{t=1}^{N} (X_t-\overline{X})^2 \quad (2.8)$$



HYPOTHETICAL TIME SERIES WITH LINEAR GROWTH TREND

FIGURE 1.



HYPOTHETICAL DETRENDED SERIES

FIGURE 2.

Hence $r_k = c_k/c_0$ is the estimator for ρ_k. In the following paragraphs we will explain, briefly, how the same acf, r_k, is used to identify tentative models for further consideration.

Nonstationary behavior is characterized by an acf which does not die out or dies out very slowly. If the nonstationarity is seasonal it will be manifested by peaks in the acf occurring s units apart where s is the period of the seasonal fluctuation. The AR(p) model is characterized by an acf which dies out exponentially or as a damped sine wave whereas for the MA(q) model the acf cuts off abruptly after lag q. The mixed model, ARMA(p,q), is identified by an acf which becomes a damped exponential and/or damped sine wave after q-p lags.

Considerably more is involved in identifying, fitting, and checking a model than simply examining the sample autocorrelation function for behavior described above, but one must begin with the acf.

2.8. Applications

A brief word on applications is in order here. In any computer system, the throughput, computer resource units consumed, I/O requests handled, etc. are not constant but fluctuate in an apparent random manner about some unknown level which is usually estimated by averaging the observations of the variable over some period of time. What we propose when we attempt to model some factor of a computer system (such as "workload") is that the random process which we observe has some unknown probability structure and that we can gain some knowledge about that structure (by estimating parameters, testing hypotheses, etc.) that will allow us to better predict the performance of the system. For example, by properly selecting the factors observed early during the operational phase one could predict the time at which the system would become saturated.

3. Analysis of Computer Systems

The Aeronautical Systems Division (ASD) at Wright-Patterson AFB, Ohio, has a computer center which, in addition to other systems, operates a CDC 6600 and a CDC Cyber 74 which we will identify as systems A and B, respectively. In order to demonstrate the methods previously described we analyzed the workload of both systems using computer resource units (CRU's) as the measurement unit. In order to reduce the variance and to avoid ill-conditioning problems we actually analyzed the natural logarithms of CRU's. This transformation preserves the relationships between successive days but makes the numerical values more manageable.

The time series that we analyzed for each computer system is CRU's for successive days of operation. Days for which there was no data because the system was not operating were deleted. There were some days in which both systems were down (holidays, etc.) and days in which only one system was down. Because we assume stationarity (which in this case is a very reasonable assumption) and are concerned with the structure of the underlying process, we are freed from tying a particular observation to a particular calendar day and can consider consecutive days of operation. There are 336 nonzero days in the System A time series and 332 nonzero days for System B. The June 18, 1977, day file for System A was destroyed. In this case of missing data one can choose to do one of four things:

(1) ignore the missing data,

(2) replace the data with an average of adjacent observations,

(3) replace the data with the ensemble average, or

(4) forecast ahead (or behind) using the data on hand to predict the actual value and use the forecast to replace the missing data.

We chose to replace the missing observation with the average of the adjacent observations. In this case it was very close to the ensemble average (11.29186 inserted, $\overline{X} = 11.21825$ without inserted point).

3.1. Analysis of System B (CDC Cyber 74)

We will present the analysis of System B first since the analysis was somewhat easier. Table I contains the sample mean and variance for System B as well as the sample acf for both the undifferenced series X_t and the series $\nabla_7 X_t$. Figures 3-5 are plots of the data and the sample acf's for both undifferenced and differenced series. An examination of the acf for X_t shows peaks near lags of 7 and 14. This indicates the possibility of a seven day cycle. After taking the seasonal difference of period 7 we see that all of the sample autocorrelations are near zero except for lag 7. This indicates the model is of the form

Table 1

Estimated Autocorrelations of Undifferenced and Differenced Logged CYBER Data

($\bar{X}$ = 11.1236 s^2 = .106)

	Lags	Autocorrelations						
(a) X	1-7	.1579	.0849	-.0447	.0259	.0489	.1680	.1914
	8-14	.0361	-.0258	-.0819	.0910	.0836	.1999	.1277
	15-21	.0402	-.0041	-.0342	.0834	.0946	.1243	.0621
	22-25	-.0599	-.0287	-.0024	.0178			
(b) $\nabla_7 X$	1-7	.0403	.0497	-.0103	.0153	-.0479	.0192	-.4318
	8-14	.0444	-.0237	-.0576	.0349	.0055	.0711	.0138
	15-21	.0454	.0115	.0027	.0332	.0603	-.0015	.0121
	22-25	.0385	-.0026	-.0147	-.0232			

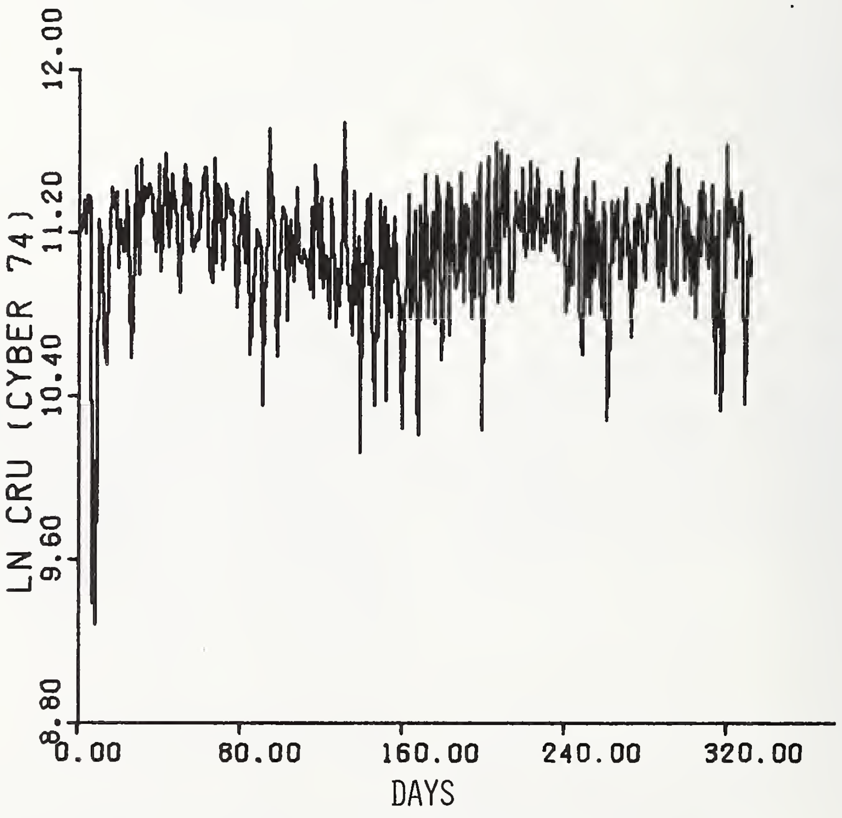


Figure 3. Logged CYBER Data

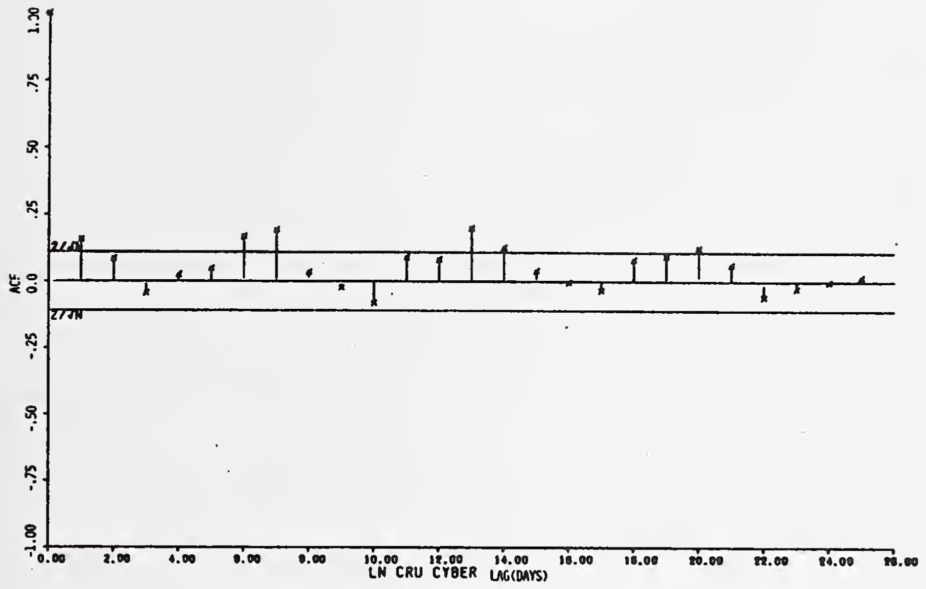


Figure 4. Autocorrelation Function for Logged CYBER Data

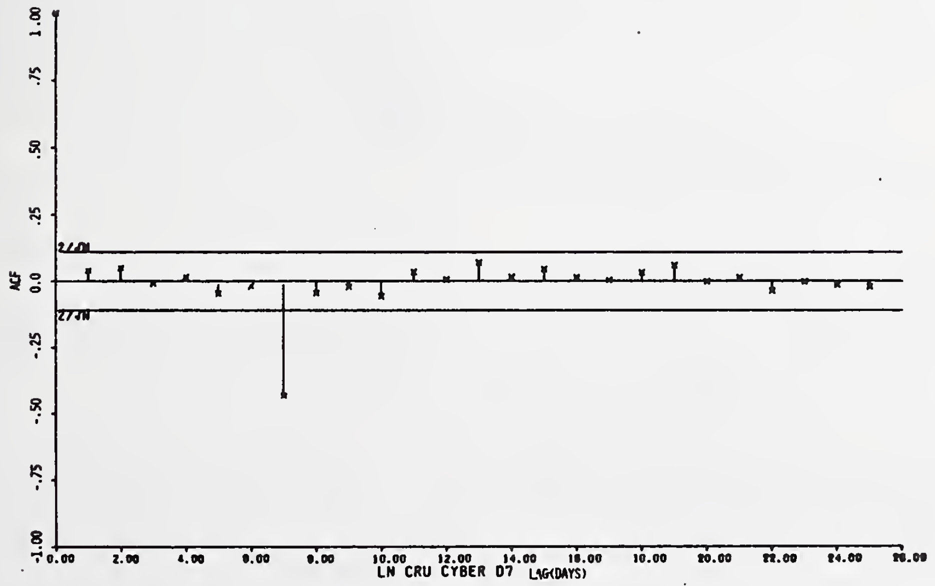


Figure 5. Autocorrelation Function for Differenced Logged CYBER Data

$$X_t = X_{t-7} + Z_t - \Theta Z_{t-7} \qquad (3.1)$$

which is a seasonal model with P = 0, Q = 1, D = 1, p = 0, q = 0, d = 0, s = 7.

Using a search method to minimize the sums of squares function on the residuals $\{Z_t\}$ we estimated Θ as $\hat{\Theta} = 0.6144$. It should be pointed out that in many cases the sums of squares surface is rather flat near Θ and small changes in $\hat{\Theta}$ may make very little difference in the total sums of squares. Hence the model will still provide relatively good forecasts even if the algorithm is stopped before attaining the actual minimum. The variance of $\hat{\Theta}$ is approximately $N^{-1}\sqrt{1-\hat{\Theta}^2}$ so for the sample size 325 (remember 7 were lost in differencing) and $\hat{\Theta} = .6144$ we get $\text{Var}(\hat{\Theta}) \approx .0019$ and the standard error for $\hat{\Theta}$ is approximately .0438. Hence an approximate 95% confidence interval for Θ is $\hat{\Theta} \pm 2\ \text{S.E.}(\hat{\Theta}) = .6144 \pm .0875$.

3.2. Diagnostic Checks

It is not sufficient to simply identify a plausible model and then estimate the parameters of the model. One must perform diagnostic checks on the estimated residuals $\{\hat{Z}_t\}$. It can be shown that if the model is adequate then

$$\hat{Z}_t = Z_t + 0(1/\sqrt{N}) \qquad (3.2)$$

so that as the sample size gets large the residuals approach the white noise sequence $\{Z_t\}$.

In order to test the adequacy of the model we will use the fact that for any ARIMA(p,d,q) process

$$Q = N \sum_{k=1}^{K} r_k^{\,2}(\hat{Z}) \qquad (3.3)$$

has approximately a chi-square distribution with K-p-q degrees of freedom, and large values of Q indicate lack of fit.

We calculated Q = 21.59 for K = 30 and as we have proposed the $\text{ARIMA}(0,1,1)_7$ model, only one parameter was estimated so that Q has approximately a chi-square distribution with 29 d.f. The .25 quantile for a $\chi^2(29)$ is 23.6 so there is no evidence of lack of fit.

For seasonal models there is one more diagnostic check on the residuals which must be made. Since the seasonal model is periodic, it is possible for the model to pass the chi-square lack of fit test but still have unexplained periodic behavior. The cumulative periodogram check is designed to detect the power spectrum for a true white noise constant $2\sigma_z^2$ over the frequency domain 0 - 0.5 cycles (frequency = 1/period). Consequently the cumulative spectrum P(f) for white noise is a straight line from (0,0) to $(0.5,\sigma_z^2)$ so that $P(f)/\sigma_z^2$, is a straight line from (0,0) to (0.5, 1) for white noise.

The normalized cumulative periodogram is an estimate of $P(f)/\sigma_z^2$ formed by adding up correlations of Z_t with sine and cosine waves at the frequencies i/N, i = 1,2,...,N/2 where N is the number of data points, and then dividing by NS^2 where S^2 is the sample variance of the residuals. If there are unexplained periodic components they will be identified by large departures of the normalized cumulative periodogram from the straight line between (0,0) and (0.5, 1). A Kolmogorov goodness of fit test was used to determine if departures from the straight line are significant. The easiest way to accomplish this test is to plot the cumulative periodogram along with 5% bands about the perfect straight line from (0,0) to (0.5, 1). This was done and is shown in Figure 6. No evidence of lack of fit is found.

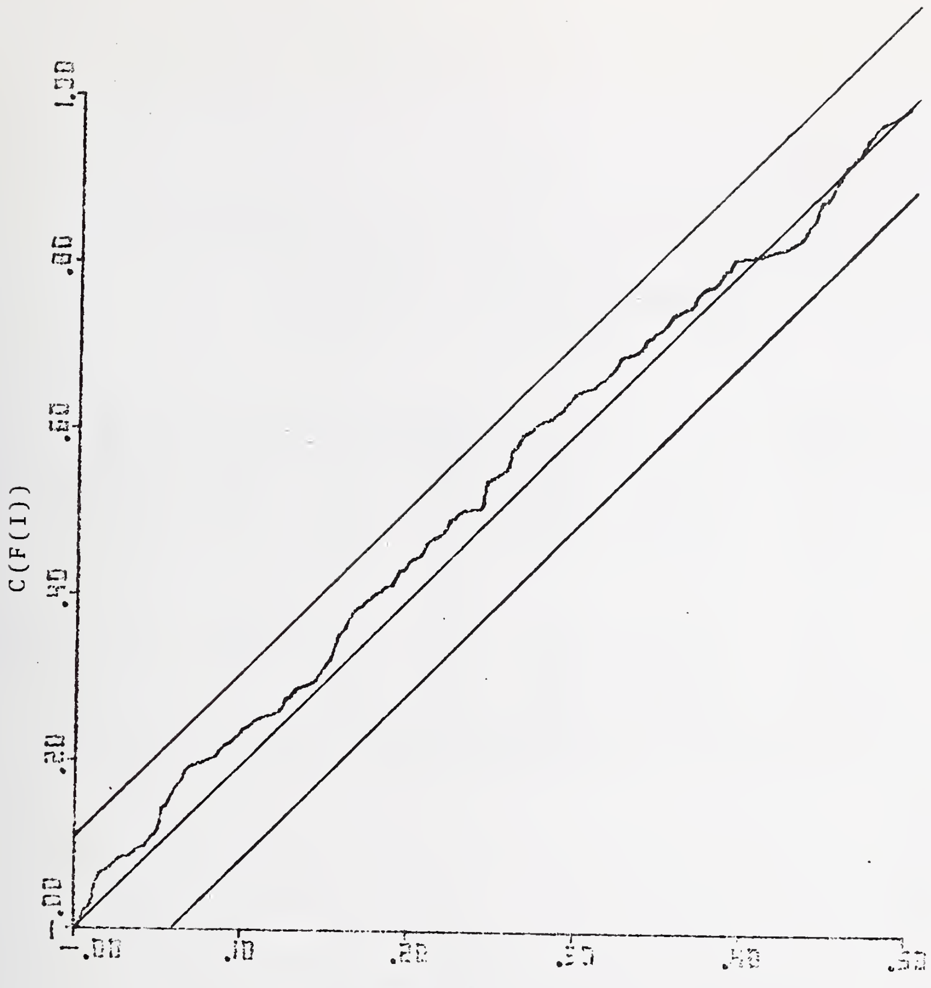
Hence we conclude that the model

$$X_t = X_{t-7} + Z_t - 0.6144Z_{t-7} \qquad (3.4)$$

is an adequate model for the logged Daily Computer Resource Units used data of the Cyber 74.

3.3. Analysis of System A (CDC 6600)

Before forecasting the expected observation for System B we will repeat the analysis for System A. The sample autocorrelation function for the data from System A is shown in Table 2 and plotted in Figures 8-9. As before we see that observations at lags 2, 7, and 13 are somewhat larger than at other lags indicative of a seasonal model. After differencing by 7 we see that all autocorrelations except for 2, 6, 7, and 13 are nearly zero and that the autocorrelation at lag 7 is much larger than the others. This suggests a model from one of the following forms:



FREQUENCY
CUMULATIVE PERIODOGRAM
SYSTEM B (CDC CYBER 74)

Figure 6.

MODEL III: $X_t - X_{t-7} = Z_t - \Theta_1 Z_{t-7}$. (3.7)

Model II was discarded after preliminary estimation procedures showed that θ_2 was essentially zero ($\hat{\theta}_2 = .0029$). Model I was entertained somewhat further. A search was made in the (θ_1, Θ_1) plane to locate (numerically) the minimum of the sums of squares of the residuals. In this case the least squares estimate for θ_1 was also very nearly zero ($\hat{\theta}_1 = .0050$). Finally we turned to Model III which is the same form as that for System B. The least squares estimate for Θ_1 is $\hat{\Theta}_1 = .69$ with standard error of 0.0399 so that an approximate 95% confidence interval for Θ_1 is .69 ± 0.0798. The χ^2 goodness of fit test for this model yielded a test statistic value $\chi^2 = 24.26$ which when compared to a χ^2 with 29 d.f. shows excellent fit (approximate level of significance is 70%). We also computed the cumulative periodogram for the residuals which is plotted in Figure 10. The straight line running from (0,0) to (0.5, 1) represents the cumulative spectrum for white noise and the upper and lower lines are the 5% bounds for the Kolmogorov goodness of fit test.

3.4. Forecasts

Because both of our models are moving average models in the seasonal difference operator we will have forecasts for the first seven days past the end of the data based

Table 2

<u>Estimated Autocorrelations of Undifferenced and Differenced Logged CYBER Data</u>

$(\bar{X} = 11.2185 \quad s^2 = 0.1014)$

		Lags	Autocorrelations						
(a)	X	1-7	.0430	.1998	-.0358	-.0197	.0888	-.0196	.1996
		8-14	-.0454	.0317	-.0274	.0207	.0250	.1226	.0688
		15-21	.0146	.0640	-.0372	.0622	.0225	.0713	-.0040
		22-25	-.0275	.0372	-.0402	.0199			
(b)	$\nabla_7 X$	1-7	.0795	.1340	-.0050	.0106	-.0141	-.1376	-.3901
		8-14	-.0716	-.0944	-.0154	-.0160	-.0549	.1208	-.0307
		15-21	.0516	.0381	-.0147	.0632	-.0037	-.0195	-.0601
		22-25	-.0598	-.0159	.0164	-.0634			

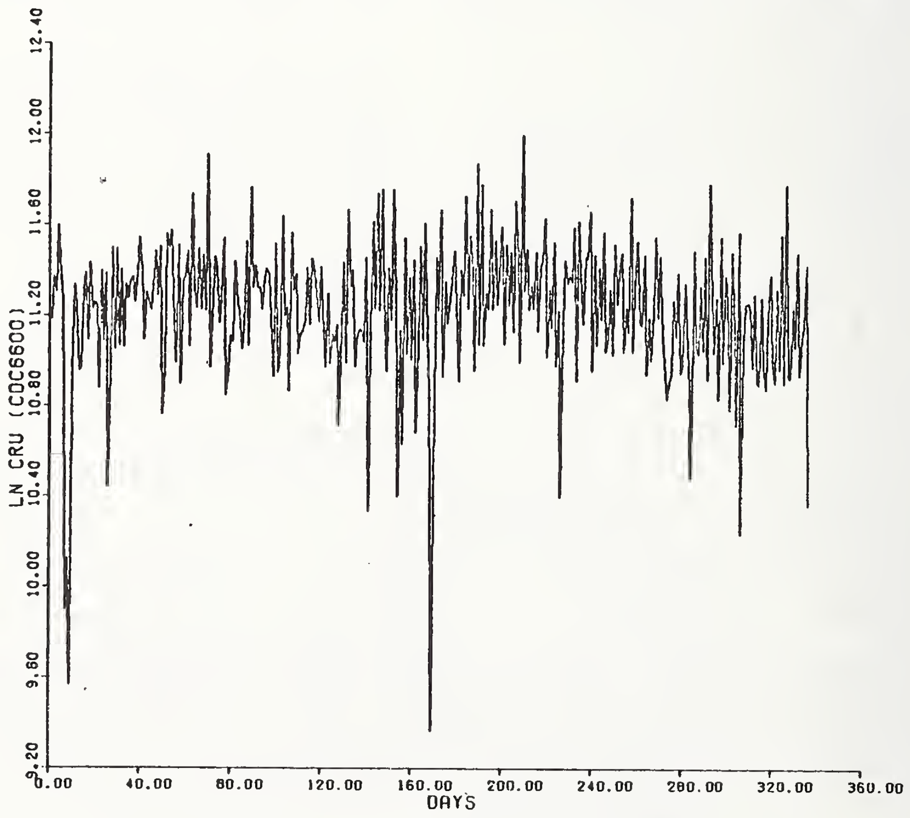


Figure 7. Logged CDC 6600 CRU Data

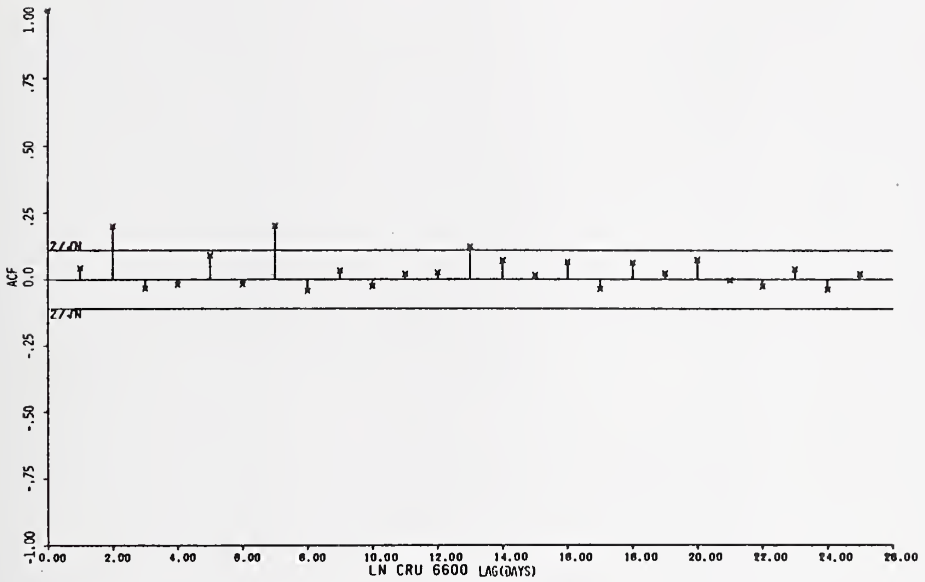


Figure 8. Autocorrelation Function for Logged 6600 Data

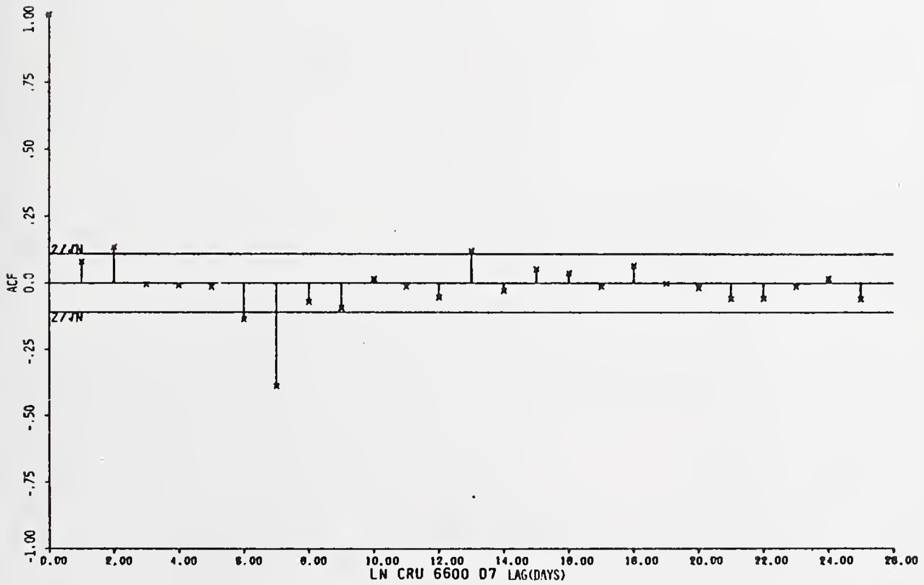
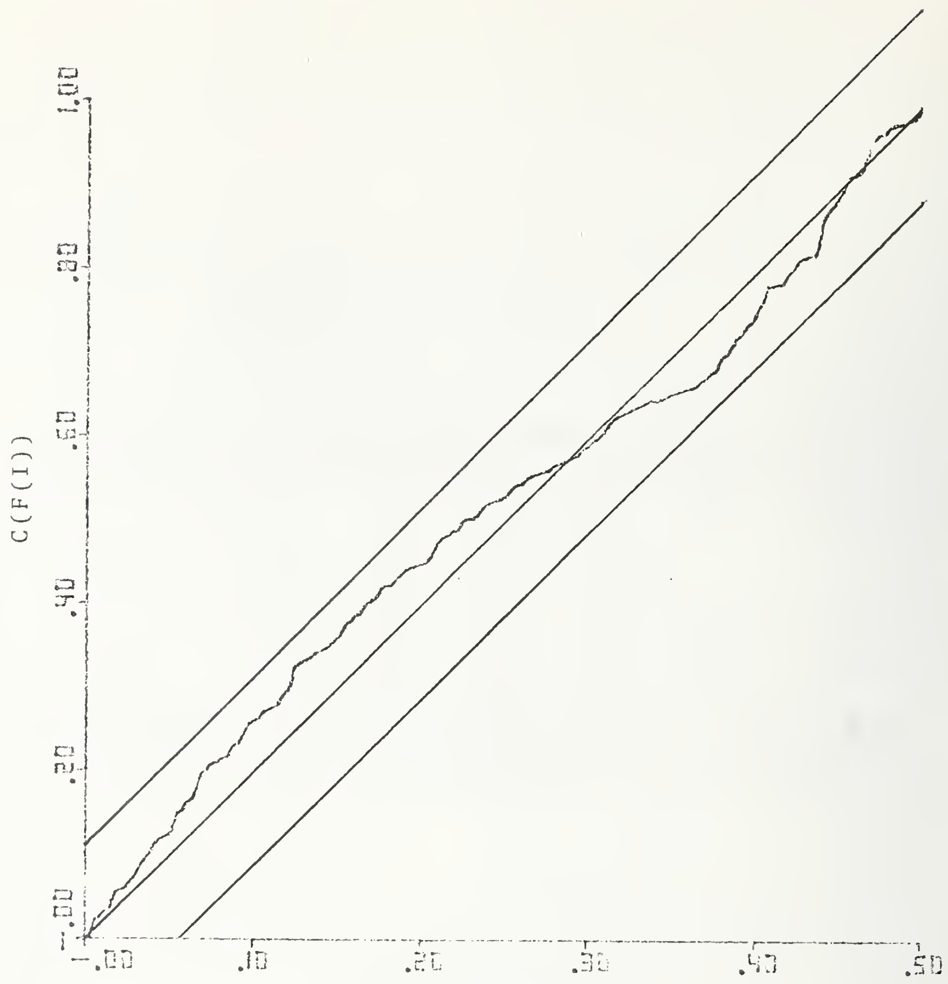


Figure 9. Autocorrelation Function for Differenced Logged 6600 Data



CUMULATIVE PERIODOGRAM
SYSTEM A (CDC 6600)

Figure 10.

upon the values and shocks on the last 7 days of data. The forecasts will then repeat themselves as the expected value of a shock is zero so that

$$\begin{aligned} E(X_{t+\ell}) &= E(X_{t+\ell-7} + Z_{t+\ell} - \Theta_1 Z_{y+\ell-7}) \\ &= E(X_{t+\ell-7}) + 0 - \Theta_1 \cdot 0 \\ &= E(X_{t+\ell-7}) \qquad (3.8) \end{aligned}$$

Additionally this type model has the characteristic that once a forecast is too high (low) it tends to remain too high (low). Thus if the forecast observations are "close" to the actual observations and the pattern is faithfully reproduced then the model can be considered "correct". In this case the best procedure would be to adjust future forecasts for the deviation between the actual obtained and the forecast. This procedure of updating forecasts is explained in Chapter 5 of Box and Jenkins [2].

The forecast and actual values for the first 21 days of operation in 1978 for the CYBER (System B) and the CDC 6600 (System A) are shown in Tables 3 and 4 and are plotted in Figures 11 and 12, respectively. An examination of Table 3 and Figure 11 shows that the pattern in the actual values obtained in 1978 is faithfully reproduced and that except for one point (19 Jan 78) the forecast value exceeds the actual value. Moreover all but two of the forecasts lie within the 2 standard error band. On the other hand, the fitted model for the CDC 6600 is not nearly as good. The pattern is not reproduced and

Table 3. Forecast Value vs Actual Value for Transformed Cyber 74 CRU's - 3 Jan 78 to 23 Jan 78

Date	Forecast	Actual	Difference	±2S.E.
3 Jan 78	10.9742	10.8769	.0973	.6991
4 Jan 78	11.6731	11.0916	.5815	.6991
5 Jan 78	11.0591	10.9435	.1156	.6991
6 Jan 78	11.9181	11.4633	.4548	.6991
7 Jan 78	11.4382	11.2993	.1389	.6991
8 Jan 78	11.2267	10.9957	.2310	.6991
9 Jan 78	10.9801	10.8585	.1216	.6991
10 Jan 78	10.9742	10.8550	.1192	.7792
11 Jan 78	11.6731	11.6106	.0625	.7792
12 Jan 78	11.0591	10.9850	.0741	.7792
13 Jan 78	11.9181	11.4147	.5034	.7792
14 Jan 78	11.4382	11.2742	.1640	.7792
15 Jan 78	11.2267	10.8095	.4172	.7792
16 Jan 78	10.9801	10.7679	.2122	.7792
*17 Jan 78	10.9742	9.8217	1.1525	.8518
18 Jan 78	11.6731	11.0783	.5948	.8518
19 Jan 78	11.0591	11.1241	-.0650	.8518
*20 Jan 78	11.9181	11.0001	.9180	.8518
21 Jan 78	11.4382	10.9427	.4955	.8518
22 Jan 78	11.2267	10.7210	.5057	.8518
23 Jan 78	10.9801	10.9012	.0789	.8518

*Outside ±2S.E. band.

Table 4. Forecast Value vs Actual Value for Transformed CDC 6600 CRU's - 3 Jan 78 to 23 Jan 78

Date	Forecast	Actual	Difference	±2S.E.
3 Jan 78	10.9875	11.1419	-.1544	.6847
4 Jan 78	12.0013	11.4692	.5321	.6847
5 Jan 78	10.8084	11.1663	-.3579	.6847
6 Jan 78	10.6237	11.1802	-.5565	.6847
7 Jan 78	11.8162	11.4779	.3383	.6847
8 Jan 78	11.0079	11.0771	-.0692	.6847
9 Jan 78	11.3677	10.8000	.5677	.6847
10 Jan 78	10.9875	11.1877	-.2002	.7169
*11 Jan 78	12.0013	11.1103	.8910	.7169
12 Jan 78	10.8084	11.3636	-.5552	.7169
13 Jan 78	10.6237	11.1006	-.4769	.7169
*14 Jan 78	11.8162	11.0840	.7322	.7169
15 Jan 78	11.0079	10.5350	.4729	.7169
16 Jan 78	11.3677	10.9417	.4260	.7169
*17 Jan 78	10.9875	10.0912	.8963	.7478
*18 Jan 78	12.0013	10.8127	1.1886	.7478
19 Jan 78	10.8084	10.2624	.5460	.7478
20 Jan 78	10.6237	10.7909	-.1672	.7478
*21 Jan 78	11.8162	10.9160	.9002	.7478
22 Jan 78	11.0079	10.7589	.2490	.7478
23 Jan 78	11.3677	11.0128	.3549	.7478

*Outside ±2S.E. band.

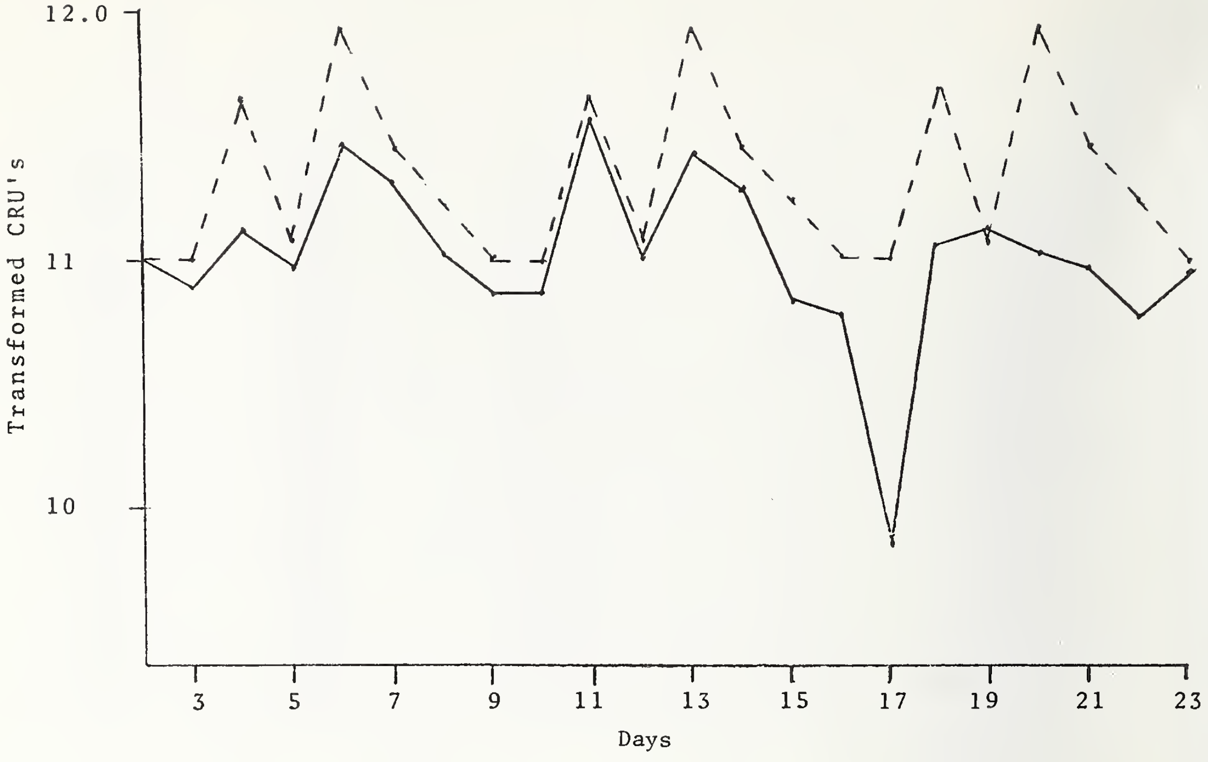


Figure 11. Forecast vs Actual CDC CYBER 74 CRU Data for 1979

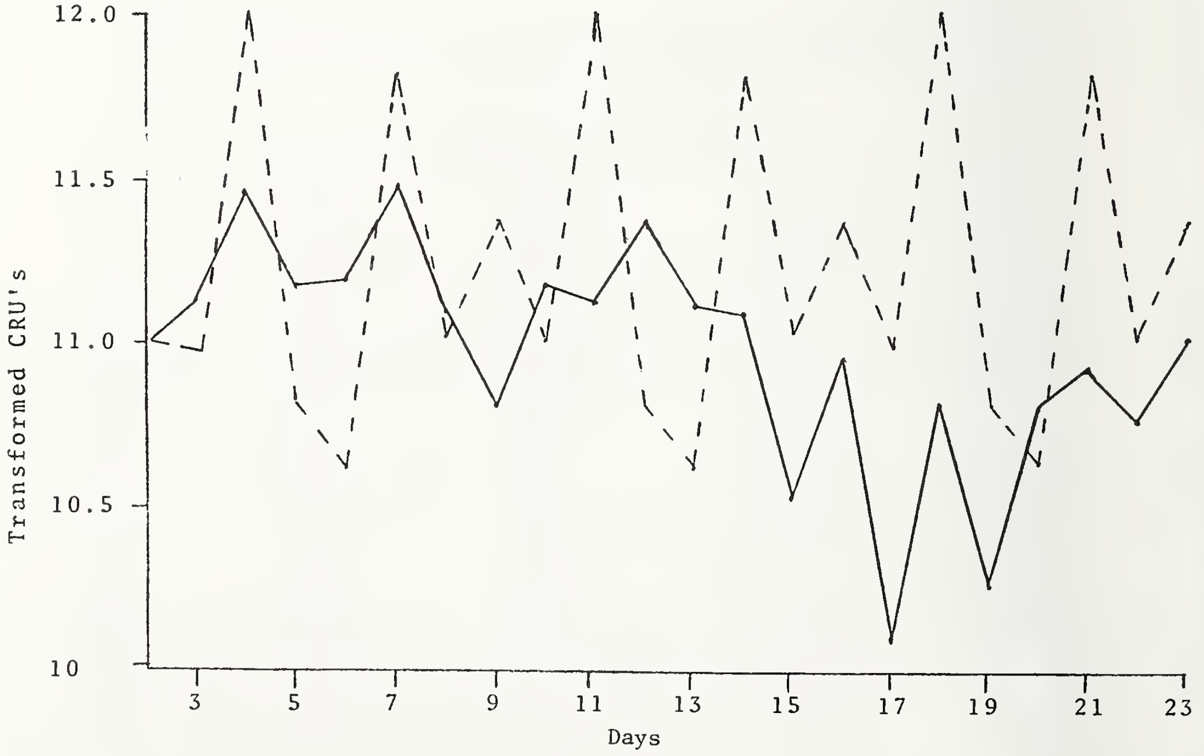


Figure 12. Forecast vs Actual CDC 6600 CRU Data for 1979

there is greater deviation between forecast and actual value. In order to explain the poor fit we decided to look at how each system was utilized.

The CDC 6600, which is used by ASD organizations, has a very heavy management oriented workload while the CYBER 74 is used mostly by the Air Force Wright Aeronautical Laboratories and the Air Force Institute of Technology. As a result of these differences it turns out that day of the week (Monday, Tuesday, etc.) is an important factor on the CDC 6600, because of the large amount of production which is keyed to the day of the week (i.e., management documents due on Monday, say). Thus it does not suffice to forecast successive days of operation for System A when the computer center is closed for an extended period as was the case for our analysis. (The last data point in 1977 was 24 December and the first data point for 1978 was 3 January). To handle this, we forecast ahead for the days that the center was closed and picked up the actual data on the proper day of the week in January 1978. Table 4 and Figure 12 do not reflect the days the center was closed. It is clear that the forecasts for the CYBER 74 (System B) are much better than those for the CDC 6600 (System A). It is not at all clear why this is so except for, possibly, the roles that the systems play are much different. External factors that might have played a role were two snow storms in the period from 14 January to 17 January (two inches on 14-15 January and 5 inches on 16-17 January).

4. Summary

We have seen that it is possible to model a measure of workload for a computer system and then use the model to forecast workload for future planning purposes. We have not considered the subject of adaptive forecasts where the forecast is updated as additional data is collected but such a procedure is not only possible, but desirable. We have also seen that one must consider external factors such as production cycle keyed to day of week or month in order to properly forecast workload when those factors are significant.

References

[1] Anderson, T. W., The Statistical Analysis of Time Series, Wiley, New York, 1971.

[2] Box, G. E. P. and Jenkins, G. M., Time Series Analysis Forecasting and Control, Holden-Day, San Francisco, 1970.

ESTIMATION OF RUN TIMES USING SIGNATURE TABLE ANALYSIS[1]

S. A. Mamrak and P. D. Amer

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Algorithms for managing jobstreams in a complex computer environment often rely on various estimates of job run times. Due to wide variability of run times from one execution of a job to another, point estimations of run times are fairly unreliable. An alternate approach to using point estimations is to use intervals which span the range of possible run time values. In an interval approach run times can be predicted with respect to membership in one of a limited set of run time intervals, with relatively high confidence. This paper presents a formal methodology for run time estimation based on an interval approach. The estimation is done using signature table analysis and is accompanied by a statement of statistical confidence in the results.

Key words: Interval estimation; point estimation; run time prediction; signature table analysis.

1. Introduction

Algorithms for managing jobstreams in a complex computer environment often rely on various estimates of job run times. Typical run times of interest include response time, processing time, turnaround time and so on. For example, scheduling algorithms which tend to minimize average job turnaround time based on the shortest-processing-time principle often rely on a prediction of what the job processing time will be. In systems which have a large degree of multiprogramming, run times for a particular job vary widely from one execution to another, depending upon the number and kinds of jobs that are simultaneously contending for resources. Prediction of run times, therefore, although fairly accurate "on the average," tends to be unreliable in any single instance because of the inherent complexity of the processing environment.

An alternate approach to using point estimations of run times, with their inevitably large variability and low confidence, is to use intervals which span the range of possible run time values. In an interval approach, run times are predicted with respect to projected membership in one of a limited set of run time intervals. The potential advantages of this technique are that in some environments prediction can be done based on very little knowledge about a job, and the confidence of predicting membership in the correct interval can be very high. The usefulness of this interval approach has long been recognized in the computer community and several ad hoc implementations exist. The classification of jobs in IBM's job preprocessor called HASP, for example, has been achieved in some installations by placing jobs in classes A, B, C and so on, based on user supplied estimates of resource requirements. Essentially, these classes represent predicted run time intervals for their respective members.

This paper presents a formal methodology for run time estimation based on an interval approach. The estimation is done using signature table analysis and is accompanied by a statement of statistical confidence in the results. It may be true that for very complex systems, subjective (or even random) estimation is the best method. This paper discusses the improvement possible on subjective "guesstimates."

2. General Background: Signature Table Analysis

Run time estimation for single computer systems is an important performance question which can be formulated in the following way: given a specified computer hardware and software configuration, and a workload which is composed of a series of jobs to be run on

[1]This research was supported by the U.S. Army Research Office under grant number DAAG29-77-G00185.

that system, what characteristics of the jobs can best be used to predict their respective run times. More specifically, let a computer system workload, W, consist of a series of n jobs, P_i, i=1, ..., n, and assume there exists a set of m descriptors d_1, d_2, ..., d_m for each P_i which characterize that job's behavior. Then, the question of interest is which subset(s) of these descriptors can be best used to predict an additional or "key" descriptor, namely run time, and what is the particular function of the critical descriptors which yields this prediction.

The nature of the run time prediction problem and the motivation for developing certain kinds of methodologies for its solution can be illustrated by placing the problem in the context of a large, production-oriented computer system. In this case, a certain number of production jobs are being run on a regular basis -- daily, weekly, monthly, and so on. These production jobs often consist of several different programs (for example, payroll runs which include not only the relevant salary calculations, but also check-writing routines and summary report routines), and require a variety of system resources. Further, due to security and deadline constraints, they are often run on a dedicated system. The production jobs are completely specified and their characteristics with respect to development, maintenance and run-time behavior, d_1, d_2, ...,d_m, can be determined in most instances. Now if a new production job, P_{n+1}, is proposed for implementation on the existing system, the specification of its required turnaround time becomes a critical factor upon which to base the decision to allow or disallow it. Some subset of the projected behavior characteristics of P_{n+1} may be known, and resources may be available to investigate others, in order to estimate the job's turnaround time. The questions of which characteristics are most important in prediction, what form the predictor should take, and with what confidence the prediction can be made, must then be addressed.

The computer run time prediction problem can be formulated in terminology that makes the application of a pattern recognition technique called signature table analysis appear extremely appropriate. In essence, this technique deals with manipulating a set of data which possesses a finite number of discrete features, as well as a "key" feature. Analyses are performed on a "training sample" for which values of all the features, including the key feature, are known. Prediction of the key feature is explored, by means of the specification of a derived (combined) feature set which approximates the key feature on the training data. The derived feature set can then be applied to other sets of data for which the key feature must be predicted.

Typically, in a run time prediction environment, a training sample or set of data is collected which consists of a finite number of workload characteristics, like CPU, I/O and core resource requirements, along with the known turnaround time of already existing production jobs. Turnaround time prediction may then be conceptualized as the problem of identifying the significant "features" among the d_i which best describes a job's turnaround time "pattern".

The signature table method of pattern recognition suggested by Samuel [SAM67] for use in machine learning problems, and further developed by Page [PAG75] is a hierarchical approach for the recognition of patterns which are described in terms of many features. The method provides a means by which features are exhaustively analyzed in subsets, each of which provides a derived feature. The derived features are combined to result in higher derived features which depend in a nonlinear manner on all of the original features. An example of the tabular structure which may result from applying the method to four features is shown in Figure 1. (Figure 1 is discussed in more detail below.)

The major advantages of the signature table method over other prediction techniques, and those that render it especially applicable to the run time prediction problem are:

1) the quality of prediction is improved as more independent features or descriptors are used (this is in contrast to some techniques possessing the counter-intuitive property that for a finite-sized training sample there is an optimal number of features),

2) it provides a natural way to deal with missing data,

3) it allows the analyst to introduce personal knowledge and intuition about the system into the calculation process (this capability may greatly reduce the amount of computation required; it is comparable to the analyst's capability in the design of fractional factorial experiments to indicate which variable interactions are important and which are not),

Figure 1. Signature Tables for One Combination of Four Binary Features

Table D_{12}

d_1	d_2	$\rightarrow$	Derived D_{12}
0	0		0
0	1	$f_{12} \rightarrow$	1
1	0		1
1	1		1

Table D_{34}

d_3	d_4	$\rightarrow$	Derived D_{34}
0	0		1
0	1	$f_{34} \rightarrow$	0
1	0		1
1	1		1

Table D_{1234}

D_{12}	D_{34}	$\rightarrow$	Derived D_{1234}
0	0		0
0	1	$f_{1234} \rightarrow$	1
1	0		1
1	1		0

$$D_{12} = d_1 + d_2$$

$$D_{34} = d_3 + \bar{d}_4$$

$$D_{1234} = \bar{d}_1\bar{d}_2 d_3 + \bar{d}_1\bar{d}_2\bar{d}_4 + d_1\bar{d}_3 d_4 + d_2\bar{d}_3 d_4$$

4) in many cases it provides better prediction than multiple regression, at less cost; this is true in part due to the use of an interval estimate approach rather than a point estimate approach as previously discussed, and

5) it is applicable to data in all formats: numeric, symbolic, ordinal and graphical.

The heuristics developed by Page to implement this technique have been specified for binary (i.e., two-valued) feature values, and therefore for the recognition of binary patterns. Essentially, the methodology requires the following steps: First,

1) Determine the appropriate predictor features.

2) Determine the appropriate cutpoint of each feature, using measures of minimum entropy (information loss) and maximum prediction. Cut-points are needed to discretize continuous features and to manipulate the allowed number of discrete values of each feature.

Then, iteratively, at each (derived) feature level, until a single derived feature is obtained,

3) Determine feature subsets or "signature types" upon which derived features are to be based. In Figure 1, for example, features d_1 and d_2 are combined to derive feature D_{12}, and features d_3 and d_4 are combined to derive feature D_{34}. But various other combinations are possible.

4) Define the derived feature resulting from the respective combined features, using an appropriate quantization method. This method is symbolized by the f_i's in Figure 1.

Finally,

5) Extract the relationship between the original features and the derived feature as Boolean expressions which describe, with some known probability, relations inherent in the data. This process is illustrated in Figure 1 for the derived feature D_{1234}. The potential usefulness of such an expression in run time prediction becomes apparent when one observes that only any three of the four feature values need be obtained to

calculate the D_{1234} value. Hence, the signature table method is a way of using the data to evolve switching functions which discriminate between members of various classes (or binary values of the key feature in Page's work).

3. A Sample Application

The signature table analysis approach was used to solve a run time prediction problem for the U. S. Army. A description of the application of the method to the Army data will serve to demonstrate its basic simplicity and its effectiveness in achieving the objective of interval estimation of run time with a relatively high degree of confidence.

The purpose of the Army study was to develop a predictor for the total turnaround (TA) time of a proposed application (production) job, based on a set of projected job resource requirements. Data for the development of the predictor consisted of 412 observations on currently running production jobs. A single observation was provided in the form of a 5-tuple, V = (CPU time, turnaround time, punch I/O, tape I/O, disk I/O). The data were divided into a training sample of 284 observations and a test sample of 128 observations.

The experiment was broken up into 4 steps: 1) division of the key feature (turnaround time) into intervals, 2) cutpoint specification for predictor features, 3) computation of derived features and 4) analysis of the results. These steps are discussed in turn below.

3.1. Division of the Key Features into Intervals

The key feature, turnaround time, was divided into three intervals: less than 10 minutes, 10-20 minutes and more than 20 minutes. These intervals appeared to be natural divisions in the data and were not chosen based on any statistical considerations of appropriateness. Also, they seemed to be reasonable intervals for use in the decision making process which would follow turnaround time prediction--namely, whether or not to allow the proposed job to be developed and supported.

The three intervals were defined by two cutpoints, 600 seconds (10 minutes) and 1200 seconds (20 minutes). For experimental purposes each of these cutpoints was investigated in a separate stage. First, boolean functions were derived to estimate if a job's run time would be less than or greater than 600 seconds. Then another set of functions was derived to estimate if a job's run time would be less than or greater than 1200 seconds. The functions which predicted with the highest accuracy from each set, based on the training sample data, were then combined to derive a single function. As will be described later in the analysis of the results, this single function was used to predict into which of the three intervals a job's turnaround time fell.

3.2. Cutpoint Specification for Predictor Features

Each of the predictor features was discretized into two ranges for each of the two experimental stages, a "low" and a "high" range. All of the predictor features were positively correlated with the key feature in that a low predictor feature value 'predicted' a low turnaround time and a high predictor feature value 'predicted' a high turnaround time. Given a particular key feature cutpoint, the predictor feature cutpoints were chosen so as to minimize the total number of incorrect key feature predictions. Cutpoints were determined using the Statistical Package for the Social Sciences (SPSS) [NIE75]. Basically, frequency tables of the form shown in Figure 2 were computed for different possible predictor feature cutpoints. The value for which (b+c) was minimized was selected as the cutpoint value. Table 1 contains the cutpoints which were computed for the two stages of the experiment. Also tabulated are the number and percentage of the 284 training sample observations which were incorrectly predicted using each cutpoint. Note that even the best cutpoint value in certain cases resulted in a large percentage of incorrect key feature predictions. This is due to a predictor feature's inability to singlehandedly forecast the pattern of job turnaround time.

3.3. Computation of Derived Features

The computation of derived features has been described and analyzed in [PAG75]. A description of the steps followed in this study will be provided here. In general, predictor features are combined to produce second level derived features. These in turn are combined to produce higher level features. The process terminates when enough of the original predictor features have been used to produce higher level features which can predict the key feature's interval value with a high degree of accuracy.

Figure 2. Derivation of Predictor Feature Cutpoints

		TA Time	
		low	high
Predictor Feature	low	a	b
	high	c	d

Table 1. Predictor Feature Cutpoint Values

	TA Time Cutpoint: 600 Seconds			TA Time Cutpoint: 1200 Seconds		
Feature	Feature Cutpoint	Number of Incorrect Predictions	% Incorrectly Predicted	Feature Cutpoint	Number of Incorrect Predictions	% Incorrectly Predicted
V1 CPU Time	60.0	25	8.8%	160.0	44	15.5%
V3 Punch I/O	1.0	123	43.3%	20.0	108	38.0%
V4 Tape I/O	400.0	53	18.7%	3110.0	57	20.1%
V5 Disk I/O	1.0	28	9.9%	1600.0	128	45.1%

Predictor features were combined in pairs. Since each feature had been divided into a low and high range by a feature cutpoint, there were four possible combinations: low:low, low:high, high:low and high:high. For purposes of computational ease, low was represented by 0 and high by 1. Once again using SPSS, frequency tables were computed to determine how many high and low key feature values existed in the training sample for each combination. For each combination the proportion of high values of the key feature, p_{high}, was compared to the proportion of high values of the key feature in the entire training sample. If the first proportion was larger, then it was judged that that combination predicted a high key feature value; otherwise, a low key feature value was predicted.

Two examples of derived features, one for the turnaround time cutpoint of 600 seconds and one for the turnaround time cutpoint of 1200 seconds, are provided in Table 2. It can be seen that a derived feature can be expressed as a boolean function or combination of the two features from which it was derived. In Table 2a, both Tape I/O and Disk I/O had to be 1 (high) for the derived feature to be 1. Consequently, the derived feature is equivalent to the boolean expression Tape I/O $\wedge$ Disk I/O, or more conveniently, V4 $\wedge$ V5. Likewise the boolean expression derived in Table 2b is CPU time, or simply V1.

Table 2. Examples of Derived Features

a. Turnaround time cutpoint is 600 seconds, $p_{high} = .845$

Tape I/O V4	Disk I/O V5	No. of observations with low TA time ($\leq$ 600)	No. of observations with high TA time (> 600)	p_{high}	Derived Feature
0	0	10	1	.091	0
0	1	18	36	.666	0
1	0	3	0	.000	0
1	1	13	203	.898	1

Boolean Expression: $V4 \wedge V5$

b. Turnaround time cutpoint is 1200 seconds, $p_{high} = .447$

CPU Time V1	Disk I/O V5	No. of observations with low TA time ($\leq$ 1200)	No. of observations with high TA time (> 1200)	p_{high}	Derived Feature
0	0	81	14	.147	0
0	1	54	8	.129	0
1	0	4	34	.895	1
1	1	18	71	.798	1

Boolean Expression: V1

The process for combining features was then repeated, this time combining the derived features. Eventaully, several final boolean expressions for both turnaround time cutpoints were determined, all of which were derived from at least three of the four original predictor features.

3.4. Analysis of Results

The final boolean expressions derived for each turnaround time cutpoint are presented in Table 3. For each expression, the number and percentage of correct and incorrect predictions have been tabulated. Based on the accuracy of prediction for the training sample, it was concluded that the variable V1 (CPU time) was the best predictor of turnaround time for both cutpoints. It should be remembered that the variable V1 used to predict below-above 10 minutes is slightly different from the variable V1 used to predict below-above 20 minutes inasmuch as different predictor feature cutpoints were calculated for each. The labels $V1_{600}$ and $V1_{1200}$ are employed below to differentiate between the two.

The variables $V1_{600}$ and $V1_{1200}$ were combined to derive a predictor of all three turnaround time intervals. This predictor is a set of boolean expressions based on four variable values:

1. $V1_{600} \rightarrow$ CPU time ≥ 60 seconds
2. $\overline{V1}_{600} \rightarrow$ CPU time < 60 seconds
3. $V1_{1200} \rightarrow$ CPU time ≥ 160 seconds
4. $\overline{V1}_{1200} \rightarrow$ CPU time < 160 seconds

These variables were combined to form the turnaround time predictions:

$\overline{V1}_{600} \wedge \overline{V1}_{1200} \rightarrow$ TA less than 10 minutes

$V1_{600} \wedge \overline{V1}_{1200} \rightarrow$ TA between 10 and 20 mins.

$V1_{600} \wedge V1_{1200} \rightarrow$ TA greater than 20 mins.

$\overline{V1}_{600} \wedge V1_{1200} \rightarrow$ no prediction

The last combination is contradictory since $\overline{V1}_{600}$ implies CPU time less than 60 seconds and $V1_{1200}$ implies CPU time greater than or equal to 160 seconds. This combination was defined to be an automatic incorrect prediction. (As it turned out, none of the test or training sample data had this combination, an indication of the consistency of the separately derived expressions.)

Finally the accuracy of the predictor was estimated using the set of test data. Since turnaround times were available for the test data, it was possible to get an estimate of $p_{accuracy}$, the proportion of accurate predictions using the predictor. A summary of the actual turnaround times versus the predicted turnaround times is presented in Table 4. The left-to-right diagonal cells represent correct prediction since the predicted interval is the same as the actual interval. Other cells represent incorrect predictions.

Of the 128 test values, 101 observations were correctly predicted, thereby providing an estimate of the overall predictor accuracy, $p_{accuracy}$, of .789. An approximate 95% confidence interval for $p_{accuracy}$ was calculated, using a normal approximation, to be (.718, .860). In almost all cases (98.4% of the time), the prediction was either correct or within one interval. That is, seldom did the predictor predict less than 10 minutes when the actual turnaround time was greater than 20 minutes, and vice versa.

Table 3. Accuracy of Final Boolean Expressions on Training Sample

	Correct			Incorrect		
Boolean Expression	High	Low	Total	High	Low	Total
TA Time Cutpoint: 10 minutes						
V1	233 97.1%	28 63.6%	261 91.9%	7 2.9%	16 36.4%	23 8.1%
$V1 \wedge V4 \wedge V5$	201 83.7%	32 72.7%	233 82.0%	39 16.3%	12 17.3%	51 18.0%
TA Time Cutpoint: 20 minutes						
V1	105 82.6%	135 86.0%	240 84.5%	22 17.4%	22 14.0%	44 15.5%
V4	91 71.7%	136 86.6%	227 79.9%	36 28.3%	21 13.4%	57 20.1%

Table 4. Estimated Accuracy of the Predictor

Predicted TA Time (seconds)		Actual TA Time (seconds)			
		0-600	600-1200	1200 +	Row Total
0-600		13	2	1	16
		81.3	12.5	6.3	
		86.7	4.7	1.4	
600-1200		1	34	15	50
		2.0	68.0	30.0	
		6.7	79.1	21.4	
1200 +		1	7	54	62
		1.6	11.3	87.1	
		6.7	16.3	77.1	
	Column Total	15	43	70	128

Legend

	Interval I
Interval J	a
	b
	c

a: No. of TA values predicted to fall into interval J which had actual TA values in interval I

b: % of all J interval predictions which fell into interval I

c: % of actual interval I values which were predicted to be in interval J

4. Conclusions

Due to the large variability in job run times from one execution to another, point estimates of run times are unreliable. Interval estimation of run times is a reasonable approach to obtaining run time predictions in which a higher confidence can be placed. The application of signature table analysis to the prediction of turnaround time in one particular environment has yielded a predictor that was simple to develop, is simple to use, and is accurate about 80% of the time in predicting membership in one of three turnaround time classes. Although this interval approach technique will not provide sufficient predictive power for all applications, it is appropriate for some application objectives and should be considered as a desirable alternative to less statistically sound approaches.

References

[NIE75] Nie, N., C. H. Hull, J. G. Jenkins, K. Steinbrenner, and D. H. Bent, Statistical Package for the Social Sciences, 2nd edition, McGraw-Hill, Inc., New York, 1975.

[PAG75[Page, C. V., "Heuristics for Signature Table Analysis as a Pattern Recognition Technique," Computer Science Department, Michigan State University, 1975.

[SAM67] Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers -- II -- Recent Progress," IBM Journal of Research and Development, Vol. 6, November 1967, pp. 601-617.

SENSITIVITY ANALYSIS AND THE RESPONSE SURFACE OF A SIMULATION MODEL OF A COMPUTER SYSTEM

Kenneth Melendez, Major, USAF
Alfred H. Linder, 2nd Lt, USAF

Air Force Institute of Technology
Wright-Patterson Air Force Base, Ohio 45433

Prior to experimenting using a simulation model, a sensitivity analysis of the model is necessary. The response of the model to perturbations in the input must be determined. Without a sensitivity analysis of the model, the analyst can not determine what is an acceptable level of error for input parameters, nor can the degree of confidence on the responses be established. In this paper a simulation model of an IBM 370/155 is analyzed by constructing the local response surface for the two primary measures of performance used in the model, CPU Utilization and Gain Factor. The computer system modeled is an operational system, however, sensitivity analysis can and should be a part of the validation of any simulation model of a computer system in any phase of the system life cycle.

1. Introduction

The application of discrete event digital computer simulation to the simulation of computer systems has been steadily gaining acceptance. This acceptance is due mainly to the complexity of the problem of analyzing a computer system. The lack of analytic tools whereby analysts can build models with which to study the behavior of existing or conceptual computer systems has contributed to the increased reliance upon simulation.

This increased use of simulation to study computer systems has not automatically yielded a keener insight into the behavior of the system. The technique of simulation has by its own nature precipitated a host of new problems which must be dealt with in a rigorous fashion.

A thorough understanding of the system which will be simulated is a necessity. Once the preliminary understanding has been accomplished and the objective of the simulation established the experiment must be designed. Upon completing these preliminary tasks, the task of constructing and validating the model is at hand. Not only must the simulation be coded correctly and void of unwanted features it must also be a valid model of the system.

The concept of validity as it pertains to the experiments which will be run is the general area towards which this paper is directed. Specifically, the sensitivity of the model to fluctuations in the input will be discussed. The response surface associated with the model is the subject of the following discussion. Without a sensitivity analysis of the model, the analyst can not determine what is an acceptable level of error for input parameters, nor can the degree of confidence on the output response be established.

A discrete event digital computer simulation of an IBM 370/155 has been constructed and is analyzed with respect to its sensitivity to variations in the input parameters. Section 2 contains a brief description of the system that was simulated. In Section 3 the sensitivity analysis performed is discussed and the characteristics of the response surface is presented. The last section, Section 4 presents the conclusions of this experiment and some recommendations on the role of sensitivity analysis in simulation.

2. The Simulated System

The Air Force Systems Command (AFSC) at Wright-Patterson AFB uses the System 2000 database management system which runs on an IBM 370 model 155 computer. AFSC uses this system to store contracts as they are developed from the conception phase through the completion phase, as well as during the purchasing phase of the acquired system. There are approximately 75 terminals used at several locations across the United States which provide data to the system.

This system, the Acquisition Management Information System (AMIS), was implemented to support contract administration and disbursement activities. One of the major objectives is to implement Source Data Automation at the buying activities, Air Force Plant Representative Offices, and the Air Force Contract Management Divisions; thus, providing them with an interactive capability to update and query the central database.

AFSC is interested in results obtained from batch-interactive-mix analysis performed on AMIS jobs run on the IBM 370. The AMIS jobs account for 90% of the workload on the IBM 370/155.

An analysis of the workload was accomplished and the workload was characterized using the following job workload parameters:

· Total CPU time
· Number of I/O requests
· CPU service time per request
· I/O service time per request
· Priority

and the system parameters,

· Interarrival time for jobs by class
· Maximum number of simultaneous terminal users
· Maximum number of jobs in the system.

A detailed discrete event simulation model of the job processing and scheduling was developed using SIMSCRIPT II.5. The internal input-output to disk, etc. was not modeled; however, the effect of such an operation was included in the model. Thus, the model consists basically of the central processor and the data input terminals.

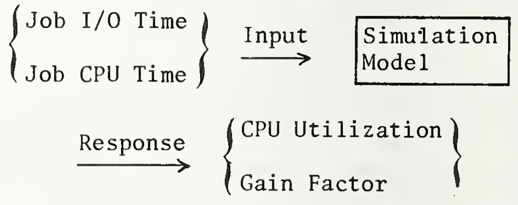
The two measures of system performance selected for use in this model are the <u>gain factor</u> and the <u>CPU utilization</u>. The gain factor is determined by finding the time needed to execute a set of jobs under multiprogramming divided by the total system time needed to execute the same jobs sequentially without the capability of multiprogramming.

Using this model of the AMIS workload and the IBM 370/155 a sensitivity analysis was performed and is described in the next section.

3. Sensitivity Analysis of the Model

The base line for the system was established assuming 28 batch jobs are scheduled during a 4 hour period and 40 interactive jobs are run during this same time. For this base line the model reflected a CPU utilization of 87% with a gain factor of 2.3684. This compares favorably with the 85% CPU utilization of the real system. The transient warm-up errors were reduced by determining when steady state conditions were achieved for the CPU utilization. This occurs approximately 5 3/4 minutes into simulation. Henceforth, statistics for all subsequent simulation runs were not accumulated until after the 5 3/4 minutes warm-up period.

Although any input parameter can be analyzed with respect to sensitivity analysis, job I/O time and job CPU time were selected for sensitivity analysis. These were the primary job workload characterization parameters subject to error. Thus, our analysis was concentrated on determining the model's response, CPU utilization and gain factor, to variations in these two input parameters.



A total of 21 runs of the simulation model were made to determine the effect of I/O time and CPU time on the cumulative CPU utilization and gain factor. The results of these runs are tabulated in Tables 1 and 2. Figure 1 is a diagonal cross section graph of the data in Tables 1 and 2 from lower left to

Table 1

<u>% CPU Utilization as a Response to % Change in Job CPU and I/O Time</u>

% Change I/O Time \ % Change CPU Time	-30	-20	-10	0	10	20	30
30	64.92						78.05
20		59.98		67.57		72.99	
10			76.19	86.42	88.73		
0		80.25	85.49	86.67	89.46	84.18	
-10			85.18	88.61	82.81		
-20		70.60		88.37		87.76	
-30	82.70						89.22

Table 2

<u>Gain Factor as a Response to % Change in Job CPU and I/O Time</u>

% Change I/O Time \ % Change CPU Time	-30	-20	-10	0	10	20	30
30	3.57						3.81
20		4.33		2.80		2.42	
10			3.10	2.63	2.72		
0		1.86	2.51	2.37	2.08	2.53	
-10			2.37	2.10	3.32		
-20		2.39		2.19		2.09	
-30	2.27						1.94

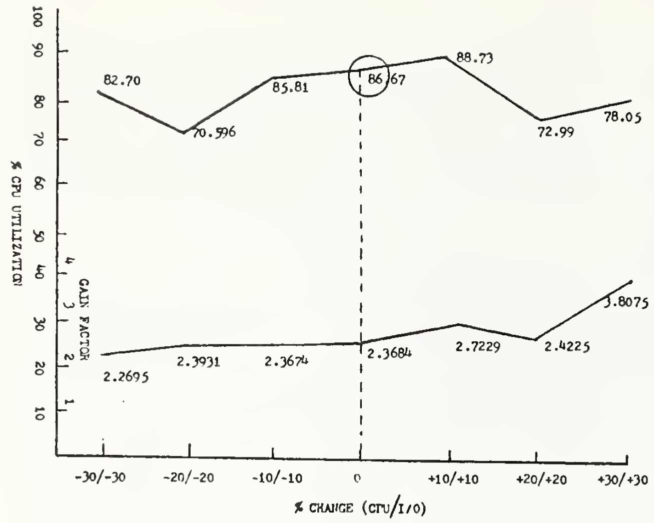


Figure 1.

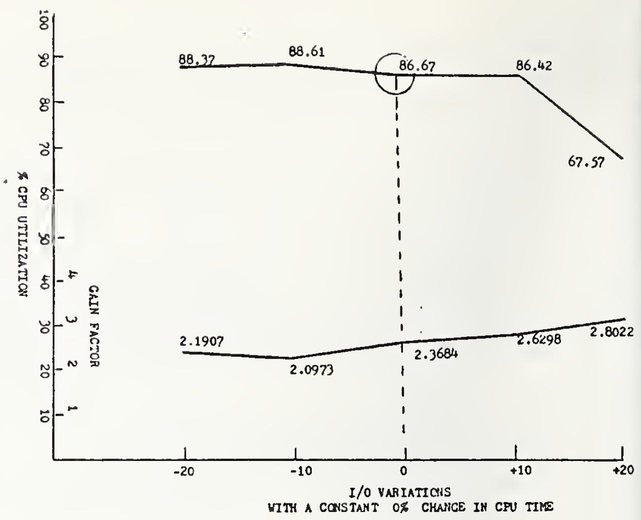


Figure 3.

upper right. Figure 2 is a diagonal cross section graph of the data in Tables 1 and 2 from lower right to upper left. Of interest is the effect on the output response to changes in just one input parameter.

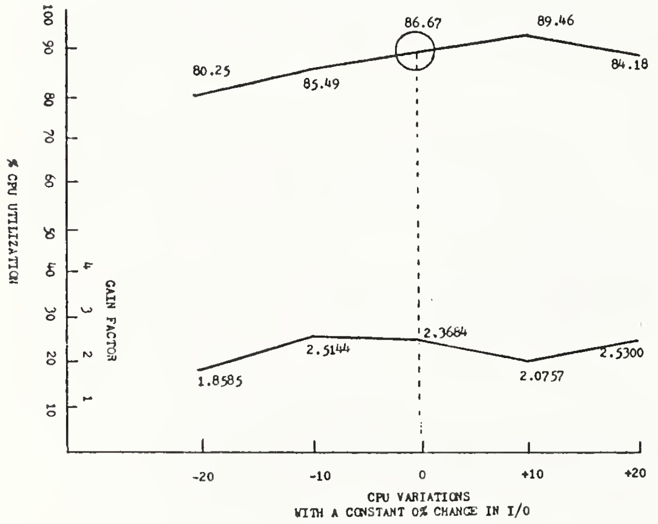


Figure 2.

Figure 3 is a plot of the response when the I/O time is held constant at 0% and the CPU time is varied. We can see that in the local area of the base line if CPU time is increased by 10% there will be a corresponding increase of 3% in CPU utilization. If CPU time is decreased by 10% the CPU utilization will decrease by 1.4%. Thus, in the local area of the base line 10% variations in job CPU time will only yield modest changes in CPU utilization of about 1.4 to 3%. We could conclude that the CPU response is relatively insensitive to variations in job CPU time in neighborhoods of the base line. Variations of 10% in job CPU time yields approximately 5.9% to 12.2% change in gain factor in the local area of the base line. Hence, we conclude that the model is sensitive with respect to gain factor when variations in job CPU time occur. The above sensitivity data was computed by using Figure 3 where a 0% change in I/O time is assumed and only CPU time changes occur.

By considering Figure 4 where the CPU time is held constant and the I/O time is varied, an analysis of output sensitivity with respect to job I/O time was accomplished.

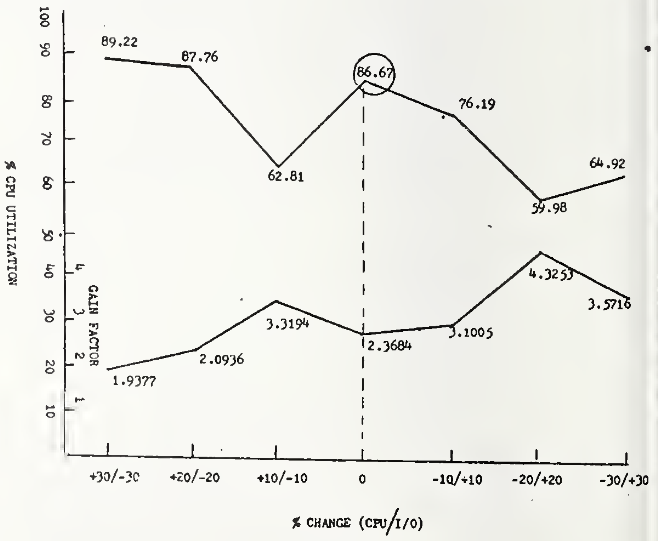


Figure 4.

Table 3

Sensitivity

	Job CPU Variations		Average Sensitivity Ratio
	-10	+10	
% Change CPU Utilization	1.4%	3.2%	.23
% Change Gain Factor	5.9%	12.2%	.91
	Job I/O Variations		**Average Sensitivity Ratio**
	-10	+10	
% Change CPU Utilization	1.9%	.25%	.108
% Change Gain Factor	11%	11%	1.10

Table 3 summarizes the sensitivity analysis for job CPU time variations and job I/O time variations. The sensitivity ratio is defined as the absolute value of the % variation in response divided by % variation in input. The average sensitivity ratio given in Table 3 is the mean of the sensitivity ratio for a -10% and a 10% change in input.

4. Conclusions

By analyzing the local response surface near the base line point the analyst can determine the relative sensitivity of the model to errors or variations in the input parameters. The sensitivity ratio as defined in the previous section will yield an indication of the relative sensitivity and can be used to identify sensitive input parameter and output response combinations. Due to the nature of the response surface, only local analysis may be performed using this technique. However, when the base line is changed a new analysis should be performed in order to determine the characteristics of the response surface at the new location.

A STATISTICAL APPROACH TO RESOURCE CONTROL IN A TIME-SHARING SYSTEM

C A Mackinder

University of Edinburgh,
Edinburgh Regional Computing Centre
Scotland

'Sed fugit interea,fugit inreparabile tempus'
Meanwhile time is flying - never to return
Virgil, Georgics III, 284

The Edinburgh DECsystem-10 Installation provides interactive computing facilities for some 200 individuals in 44 geographically dispersed groups with a wide range of research projects in science and engineering. In addition to managing resources in the sense of the instantaneous or transient demand on the machine eg: balancing the number of job slots made available and the maximum permitted job size, with the response times experienced by users, there is a need to measure and control usage over a period, say 4 weeks, both for ensuring an equitable sharing of capacity and to prevent overload. A conventional rationing system was in use but whilst it could limit the usage of the groups individually it could not be used to manage the load on the machine as a whole. A study showed that usage conforms to a gamma probability density function and is highly predictable. This has allowed the determination of an "upper limit" of usage beneath which groups are free to compute as they wish in any 4 week period, within the limit of their project's overall allocation. Groups are inhibited only if overload is threatened. The rationing system has been dispensed with. It is likely that all time-sharing systems will show similar usage characteristics and the resource control method used at Edinburgh could be applied elsewhere.

Key words: Time-sharing resource control; statistical approach to management; usage patterns; probability of size of usage; management of resources; rationing.

1. Introduction

This study arose from a desire to design a means of resource control on a large interactive time-sharing installation which would prevent the machine from being overloaded, whilst allowing users freedom of access except when overload is threatened. The method now in use has allowed a conventional rationing system to be dispensed with.

2. Background

The Edinburgh DECsystem-10 Installation is part of the Science Research Council's (SRC) Interactive Computing Facility (ICF). This includes a long-distance network run under DECNET protocols which at present connects three DEC host processors and 11 nodes. The Installation is run by the Edinburgh Regional Computing Centre, University of Edinburgh on a contract with the SRC. The Installation's equipment belongs to the SRC; the Installation's staff are employees of the University. The present configuration of the machine is given in Table 1 , but the model of the machine's capacity used in the study was based on an earlier, slightly smaller configuration. The capacity of the machine as it is now is under review.

Table 1.

Edinburgh DECsystem-10
Configuration (1070)

KI10 CPU
256K words main memory
350 Mbyte disk store
512K words fixed disk (swapping)
2xMT drives
Synchronous & asynchronous communications processors (PDP11/15 & 11/40)
Fast drum plotter
Various slow peripherals
4 Nodes connecting remote users
70 TTYs/VDUs contending for 32 job slots

Control and direction of the Installation comes from the SRC's Network Management (of the ICF), but day to day running is the responsibility of the Installation management. The Installation was set up in December 1976, although it had had an earlier existence as a central facility for several research projects.

2.1 The User Community

Users of the machine are mainly SRC funded research groups and do not pay for the resources they use. Other users pay at a 'full cost' rate if government funded (other research councils, etc) and there is a little work for which a 'commercial' rate is charged. The University itself purchases a very small amount of time at a concessional rate. In addition there are a few individuals or very small groups experimenting with interactive computing with a view to making use of the machine for potential research projects. Their usage is not paid for and for the purposes of this study it was lumped, together with some other casual and miscellaneous usage with systems usage. At the start of the period used in the study there were about 120 users in 29 groups. There are now about 210 users in 44 groups. Groups vary in size from one to about 25. While most are in Edinburgh, users may be up to 450 miles away, and the number of geographically remote users is increasing.

2.2 Type of Work

In the early stages its work was almost entirely that of its existing community of Artificial Intelligence (list processing) and Computer Aided Design (graphics) research projects. Since being taken into the SRC ICF network it has taken on a more general population of SRC Engineering Board research projects (packages) and this element of its work will continue to expand.

2.3 Control of Access and Usage

All access, and the broad control of usage, is retained by the SRC. The Installation management has a very limited discretion in allowing small occasional accesses. Allocations are made to groups by the

SRC in Allocation Units (AUs). This is an arbitrary unit which may be considered to be a week's worth of computing for one person. As will be explained later, the allocation is consumed in the form of computing resource units (CRUs) which are common units of four resources (CPU time, etc). Although AUs are used throughout this paper they can be regarded as any convenient unit (eg pounds sterling, or dollars) because the relationships between the four resources measured and AUs are linear.

Groups may be running one or more research projects. Projects last from one to three years and each has its own allocation of resources, not transferable between projects. The allocation is therefore an estimate of the computing resources needed by a group for a particular research project for up to three years ahead. It is the intention that the various "Subject Committees" of the SRC will each have a quota of AUs from which to make allocations of computing resources to groups at the same time as giving a group the funding which activates the research project. Since the committees work independently there is an inherent difficulty in achieving a more or less level loading of the machine in terms of allocations made. However, the ICF Network management advises the committees and acts as a coordinator in this respect.

For reasons of SRC grant administration the allocation is given a notional money value and forms part of the grant award. The allocation once made cannot be readily increased. It is therefore a fairly firm limitation on a group's computing activity, and because it cannot be easily adjusted there is a tendency to set it high rather than low. However, the main cause of disparity between allocation and subsequent actual usage over the life of the project is the difficulty in making a good estimate of the computing needs of a research group over a forecast period of up to three years. There is an even greater difficulty, if not an actual impossibility, in trying to make a sensible forecast of the profile of their usage, ie, in AUs per month. The need for this will be brought out later in discussing rationing (see Annexure A).

Although in general, the larger the group the larger the allocation, the allocation reflects the computing requirements of the project rather than the size. One of the largest allocations is held by a very small group.

Usage of resources is measured by the following algorithm which is derived from the theoretical model of the machine when optimally loaded, and gives a weighted sum of the CRUs used.

$$\text{CRUs} = C/0.7 + T/25 + K/28 + B/240$$

where

C = CPU time in hours

T = terminal connect(ion) time in hours

K = core occupancy in kilo word-hours (=conventional DEC kilo core secs (in thousands)/3.6)

B = disk I/O transfers in kilo blocks

CRUs are charged against a group's allocation according to the time band in which usage occurs, there being a premium on Peak time, of course.
Thus

1 CRU in Peak time	=	1.38AUs
1 " " Standard time	=	0.69AUs
1 " " Discount time	=	0.276AUs

The time bands are given in Table 2.

Table 2.
Time Bands for Differential Charging

1000-1200 1400-1800	Peak Time	Mon-Fri
1200-1400 1800-2200	Standard Time	Mon-Fri
2200-1000	Discount Time	Mon-Fri
1000-1000	Discount Time	Sat&Sun

A group can therefore get 5 times as much computing out of its allocation if it works in Discount time only, than if it works in Peak time alone. Since groups inevitably work in all three bands, forecasting the profile of their usage in AUs from month to month, which is difficult enough, entails forecasting separately how much they will use in Peak, Standard, or Discount time, which is clearly impracticable.

Obviously, Peak time is the most popular time, but the demand in Standard and Discount time is such that there is no need for any restriction on usage in these bands from the point of view of overload or of sharing resources equitably. This study therefore was concerned only with controlling usage in Peak time, but its principles could just as easily be applied, separately, to Standard time, or to Discount, in parallel systems.

2.4 Priorities

All groups have equal priority of access (ie, for job slots) and no group has special privileges within the machine in job queues, etc. The scheduler is set to handle all work purely in terms of job size, or time in queue.

2.5 The Job Mix

Practically all work generated at the terminals is interactive and only an insignificant amount is done, at present, as batch work. Batch jobs run only when the machine is idle and therefore make little progress during the day.

2.6 Records and Accounting

Accounting is by 4-weekly statistical periods (SPs) which were introduced some years before purely for their value in producing comparable information. There are thus 13 SPs to the year; they start at midnight Sun/Mon and are numbered sequentially, hence SP7809 is the 9th SP of 1978. SP7801 started on 2 Jan 78. A calendar for three years is given in Table 3 for reference but it is not important in this paper. Basic usage data in hours, kilo core-secs. etc.,is accumulated by a standard DEC accounting program FACT, and checked weekly for completeness and consistency. At the end of each SP, the data is totalled by individuals and projects within time bands and converted to CRUs and AUs. Management summaries of user activity are produced 4-weekly and usage for the machine as a whole cannot be obtained readily between times. However, at log-out each user's activity is converted to AUs and added to running totals for the project so that, at log-in, the machine can check usage to date against total allocation or any other control figure (previously the project's SP resource quota). Users can access these files for their own information, and also receive detailed 4-weekly summaries of activity for each project. It is not known whether users like statistical period accounting - but they have never voiced any objection.

3. Resource Control

Before discussing a statistical approach , it is worth making some general points about resource control and about a rationing system which was in use in the Installation,in order to explain the philosophy behind the new approach. Since much of the detail of the rationing system is not of immediate importance it has been relegated to Appendix A and can be referred to if support of the general statements, made below, is sought.

Some sort of resource control is inevitable in any large and hence expensive interactive installation.

Since it will be costly to run it should be fully utilised so this at least calls for resource control in its very broadest terms - balancing the present and known future load against some measure of the machine's capacity. Other resource control may be concerned with relieving transient or short-term shortages of CPU time or disk space.

The resource control system of the Edinburgh Installation grew out of a rationing system which, although adopted with the best of intentions, was beginning to be used for an aspect of resource control (the control of the total loading on the machine), for which it was not designed. As a result of the

Table 3
Edinburgh DECsystem-10
Statistical Periods (SPs)
In 1976, 1977 and 1978

1976	
7601	5 Jan - 1 Feb
02	2 Feb - 29 Feb
03	1 Mar - 28 Mar
04	29 Mar - 25 Apr
05	26 Apr - 23 May
06	24 May - 20 Jun
07	21 Jun - 18 Jul
08	19 Jul - 15 Aug
09	16 Aug - 12 Sep
10	13 Sep - 10 Oct
11	11 Oct - 7 Nov
12	8 Nov - 5 Dec
13	6 Dec - 2 Jan
1977	
7701	3 Jan - 30 Jan
02	31 Jan - 27 Feb
03	28 Feb - 27 Mar
04	28 Mar - 24 Apr
05	25 Apr - 22 May
06	23 May - 19 Jun
07	20 Jun - 17 Jul
08	18 Jul - 14 Aug
09	15 Aug - 11 Sep
10	12 Sep - 9 Oct
11	10 Oct - 6 Nov
12	7 Nov - 4 Dec
13	5 Dec - 1 Jan
1978	
7801	2 Jan - 29 Jan
02	30 Jan - 26 Feb
03	27 Feb - 26 Mar
04	27 Mar - 23 Apr
05	24 Apr - 21 May
06	22 May - 18 Jun
07	19 Jun - 16 Jul
08	17 Jul - 13 Aug
09	14 Aug - 10 Sep
10	11 Sep - 8 Oct
11	9 Oct - 5 Nov
12	6 Nov - 3 Dec
13	4 Dec - 31 Dec

author's study of this he is now of the opinion that unless a rationing system can be dynamic, that is, constantly forecasting future requirement against planned capacity and adjusting the "rations" accordingly, then a rationing system, especially a cautious one, is likely to hold back spare capacity rather than make it available - and in a time-sharing system, time-based resources like CPU time if held back are obviously lost forever. This is of course a generalisation, but the author has particularly in mind the kind of systems where "rations" are based on a month, or a week, or even a day, because as is shown in Annexure A the ration may be some permitted maximum and the total of all the permitted maxima is related to the theoretical or actual capacity. The permitted maximum and the actual usage are independent variables except that the former may limit the latter. Therefore it is the sum of the present or forecast actual usages which must be balanced against the total capacity in order to assess spare capacity, and the spare capacity must be made available on demand, if it is not to be wasted.

There are of course, control measures concerned with the instantaneous or transient load - the first of these is the number of job slots made available; then the machine might be said to be self-controlling by virtue of the deterioration of response time as demand increases. Priorities or privileges set into the scheduler are resource controls often concerned with favouring one user rather than another. This class of resource control is outside the scope of this paper which is concerned with control over a significant period of time, typically 4 weeks, possibly down to 1 week as a practical proposition.

3.1 The Edinburgh Rationing System (for more detail see Annexure A)

The rationing system was introduced originally to allay a fear that a research group (or groups) would, unintentionally or otherwise, under-use its bulk allocation and then make excessive demands on resources at some time possibly later in the project; or that there might be a coincidence of large demands from several projects. This overlooked the fact that some research projects might reasonably have a humped or ramp usage profile in the computing activity of their project. Some actual profiles at Peak time usage to a common scale are given in Annexure A. At the time, the machine was under- loaded, even in Peak time, so the rationing system was a control for future eventualities.

Since there was no data on the usage patterns which might be expected with the four resources being measured, the system adopted was that each project started the SP with a resource quota determined by its total allocation plus a carry forward from the unused parts of previous quotas, the carry forward decaying exponentially. On log-in by any user the total usage made by the whole group up to that time, in Peak, Standard and Discount time, was compared with the resource quota , and if the quota had been exceeded the whole group was barred from access in Peak time (only) until the next SP brought a new quota. A group had to have some unused total allocation in order to get a fresh resource quota for each SP.

Algebraically:

$$A = \alpha A' + \beta a \text{ (for } A'>0)$$

$$A = \beta a \text{ (where } A' \leqslant 0)$$

where

A was the resource quota at the start of the SP

A' was the carry forward remaining at the end of the preceding SP

a was the SP ration :

$$= \frac{\text{total allocation in AUs}}{\text{No of SPs in project}}$$

α was the shaping constant for the exponential decay of carry forward and was intended to be set between 0 and 0.5, being nearer to 0 for a large project where the overlapping demands of the individuals within it would tend to reduce the variability of its total demand.

β was the SP ration multiplier allowing higher than SP ration accesses to be made. It was usually set between 1 and 2, being larger for small allocations to allow for a higher variability in the activity of small groups.

The constants α and β catered for the circumstances that larger groups are less variable in their demand on the machine, while small groups may be more so (as a percentage of their mean activity). By definition, a rationing system involves some pro-rata spread of a total allocation over the SPs in a project, usually an even one. This might be called the "intended average rate of consumption" but clearly it is unrealistic to expect a group to work strictly at a level rate and it must therefore be allowed higher than average usages or it will not achieve the intended average over the life of its project - and may therefore be prevented from using the whole of its allocation.

The carry forward was intended to allow groups to gain some benefit from making a below average demand. The combined effect of α and β allowed a maximum resource quota of about 4 times the SP ration. A study on the machine in its pre-Installation days had shown that groups seldom demanded more than 3 times their SP ration.

Further detail of the system is not important here and it is sufficient to point out some of its shortcomings. The SP ration was the "intended average rate" of working. It was based on the project allocation and after 12 SPs it was found that the ratio:

$$\frac{\text{intended average rate}}{\text{actual average rate}}$$

which ideally should have been about 1, was anything from 1 to 30, and generally high, so that most groups were working well below their resource quota each SP. This is not so much a criticism of this particular scheme as an illustration of a fundamental weakness in any rationing scheme based on the estimation of long term requirements ("long term" being relative to this context).

The system potentially denied access to resources without regard to what was available. Although it was intended that the total of allocations made should have some relationship to the capacity of the machine, the machine was under-loaded at the time, yet a project could be barred from Peak time working for part of an SP simply because its resource quota was exhausted.

Of course groups could, and did, appeal to the Installation staff for their bar to be lifted but the staff had two major inhibitions in doing this:

(1) The immediate one was that the machine had to be given a fresh quota to match against the group's activity. The staff could give a group a generous or a miserly addition - in other words the Installation staff could assist or hinder the progress of research work insofar as this is determined by the computing resources made available. The Installation staff were not competent to make such decisions.

(2) The second one was of the future. If the eventual filling of the machine's capacity was pre-supposed the Installation staff could not know from day to day when a request was made, whether there was indeed any spare capacity to give away, and daily outputs from the accounting files were out of the question.

The rationing system was not popular with the users, not because they wanted to be profligate but because its purpose was to smooth their activity levels, or at least prevent excessive 'lumps' appearing and this threatened to restrict them "just when they wanted to get some work done".

Lastly, in theory the constants α and β would eventually be used to regulate the total use of the machine and keep this in balance with the total capacity. Annexure A shows that this cannot be realised. A load can only be regulated by observing actual usage because actual usage is a variable independent of resource quota - unless the system is so repressive that every group hits its maximum all the time.

4. A Statistical Approach to Resource Control

Thus in designing a new system the following considerations were either important or desirable:

(1) All groups to have equal priority of access (a helpful point).

(2) The actual usage a group might make next (or in any) SP could not be forecast from its allocation therefore the system had to ignore allocations.

(3) The actual usage a group might make could not be forecast without an unrealistic effort in continuous detailed study of the activity of 30 (and rising) separate groups

(4) Given that the capacity of the machine could be quantified, the system had to be able to give away any spare capacity this SP, without prejudicing the management's ability to provide for the exingencies of next SP.

(5) Although user groups disliked the limitation of a rationing scheme it was probable they would accept the notion of limitation "because the machine is full" provided they were convinced that reasonable parameters were in use - and they were given complete freedom, albeit in competition with each other, until overload was threatened.

This meant that the starting point of the control calculations had to be the capacity of the machine and the actual usage being made - from which spare capacity could be deduced. Also, since neither the allocations, nor the size of the group could be used for forecasting , the aim must be to manage the behaviour of the population as a whole.

It is necessary now to discuss the usage of the machine in terms of "accesses" which are defined for this paper as "the total usage made by all the users in any one project, in Peak, Standard or Discount time as specified, in any one SP. (In DEC terms - the total usage made by all the project-programmer numbers (PPNs) in any project in one SP). This is introduced for simplicity and to divorce the size of a group, or its allocation, from the size of the actual usage it might make in any one SP.

Table 4 shows a frequency analysis of Peak time accesses by size and SP, for the first year (13 SPs) of the Installation. The total frequencies are shown as a histogram in Figure 1. Two striking features are apparent:

(1) The great majority of accesses account for a minor part of the total usage, and

(2) The observed frequency distribution is remarkably smooth.

The first gives scope for a "Pareto" approach to the management of resources; the second suggests that a probability method can be used. It was fortunate that during the period under review, the rationing system had seldom limited a group, and when it had, the under-loading of the machine allowed the bar to be lifted. Thus the distribution is a random one reflecting only the accidents of life of the machine itself and the groups using it.

4.1 The "Pareto" Approach

Because the distribution is heavily skewed, the actual usage accounted for by each size of access had to be calculated from original data - and not from the histogram. However, 67% of the accesses were in the $<$ 5 AUs range and accounted for only 16% of the usage - while only 33% (those $\geqslant$ 5 AUs) accounted for 84% of the usage. Using the "Pareto" approach the great majority of accesses could be ignored, and provided the 33% making "large accesses" (ie $\geqslant$ 5 AUs/SP) were somehow kept in check, overload could be prevented.

4.2 The gamma probability density function

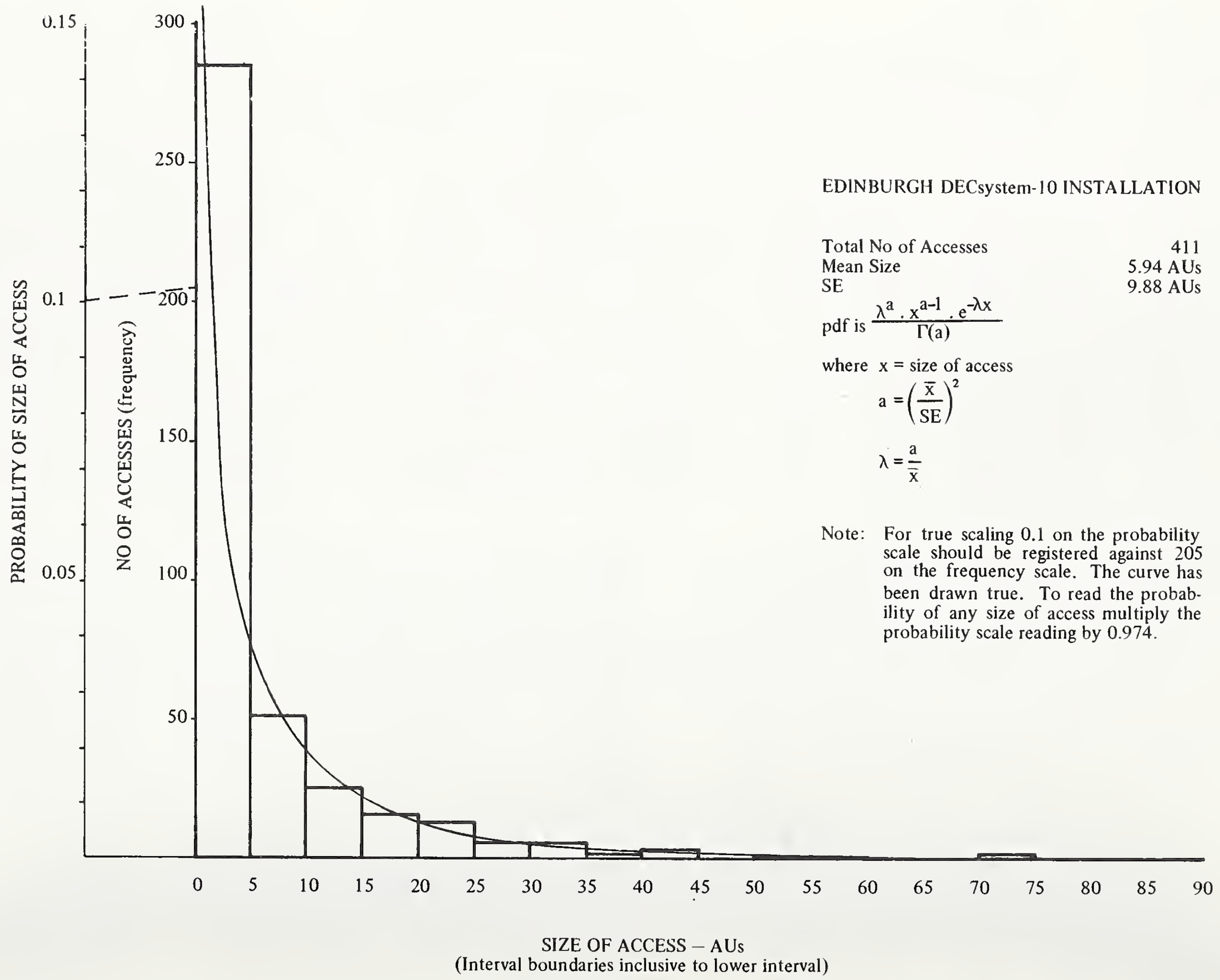
It is well known that the distribution of the length of local telephone calls conforms to a negative exponential distribution - and in considering interactive computer usage we have an obvious similarity in terminal connect time, except that the calls will tend to go on much longer. Since CPU usage and core occupancy are also time-based it is likely that they will have a similar distribution of size, and fortunately, disk I/O transfers show the same characteristics. Of course different kinds of work produce different proportions of connect time, CPU time etc. However, in the size of accesses we are therefore looking not at the distribution of a single variable but at the sum of several exponentially distributed contributing variables and the gamma probability density function (pdf) defines such a distribution. A gamma pdf is shown fitted to the histogram in Figure 1.

Table 5 and Figure 2 show the same information for a later year SPs 7707-7806 (20 Jun 77 - 18 Jun 78) which overlaps the first by 6 SPs. The number of projects on the machine has risen to 44 and the sample of accesses has lost 218 of the original observations and gained 297 giving a total of 490, so that about 2/3 are new observations. The usage of the machine by research projects has risen by about 25%. It is interesting to note that the mean size of access has not changed (in the intervening moving years it has varied only between 5.94 and 6.22 AUs). In round figures, the usage accounted for by small accesses has hardly changed - 66% of accesses; 14% of usage.

Since these frequency distributions are the outcome of from 120 to 220 users in 30 to 44 groups making 10,000-15,000 log-ins over a year it is a reasonable assumption that they are stable and can be used for forecasting based on probability.

Table 4. Analysis of Peak Time Accesses on Edinburgh DECsystem-10
By Size & Date

Size of Accesses (AUs)	Number of Accesses in Statistical Period													Total
	7613	7701	7702	7703	7704	7705	7706	7707	7708	7709	7710	7711	7712	
0-4.99	26	25	22	21	22	22	20	18	19	21	21	24	24	285
5-9.99	2	3	3	3	3	4	6	5	6	6	3	2	5	51
10-		1	1	1	2		4	2	4	1	4	3	2	25
15-			2	1		4	1	3	1	1		1	2	16
20-	1	1	1	2	2	1	1	2				2		13
25-				2	1		1				1		1	6
30-			2			2			1		1			6
35-				1						1				2
40-					1			1	1					3
45-														
50-										1				1
55-												1		1
60-														
65-														
70-											1		1	2
Totals	29	30	31	31	31	33	33	31	32	31	31	33	35	411



EDINBURGH DECsystem-10 INSTALLATION

Total No of Accesses	411
Mean Size	5.94 AUs
SE	9.88 AUs

pdf is $\frac{\lambda^a . x^{a-1} . e^{-\lambda x}}{\Gamma(a)}$

where x = size of access

$a = \left(\frac{\bar{x}}{SE}\right)^2$

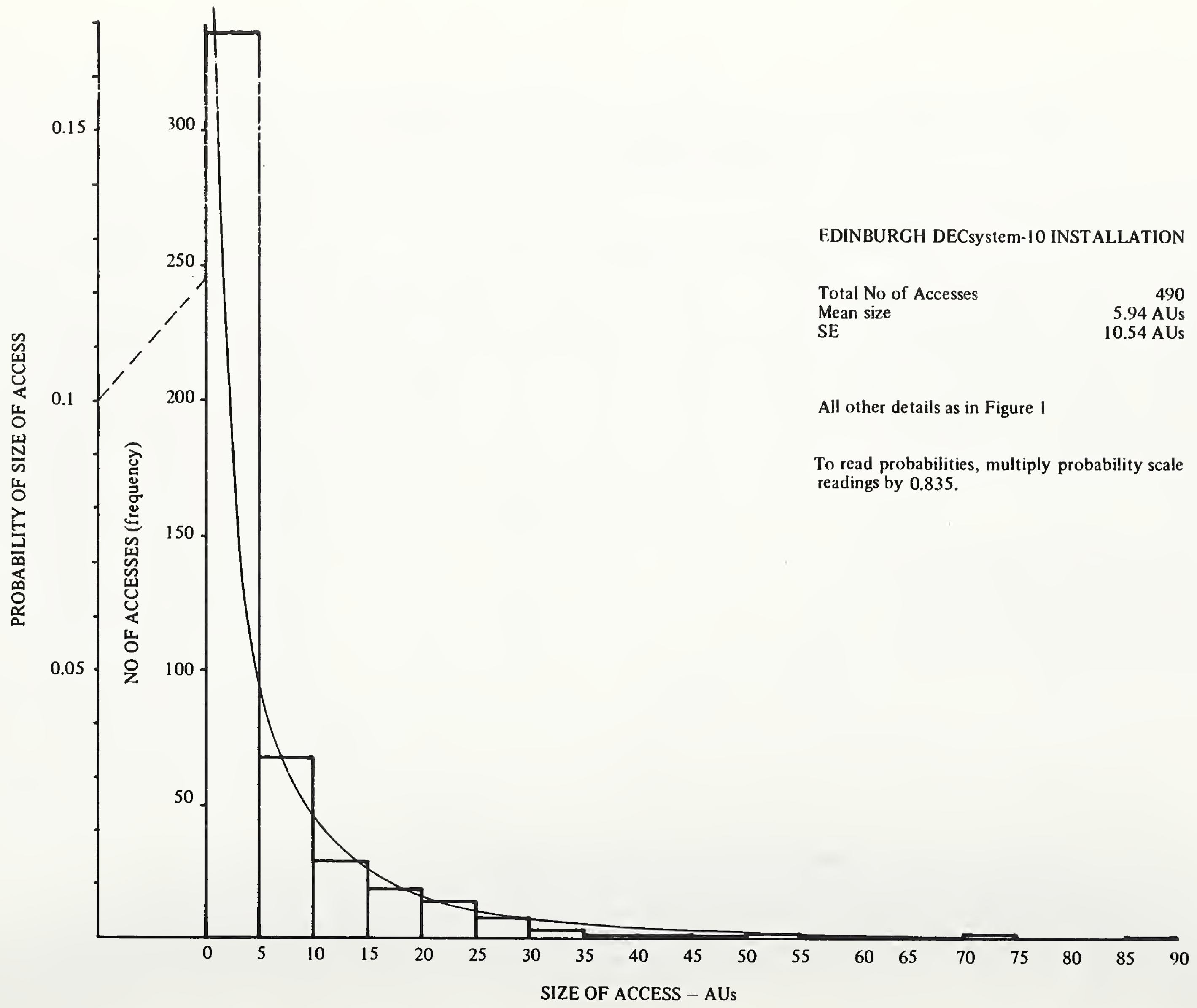
$\lambda = \frac{a}{\bar{x}}$

Note: For true scaling 0.1 on the probability scale should be registered against 205 on the frequency scale. The curve has been drawn true. To read the probability of any size of access multiply the probability scale reading by 0.974.

Figure 1. Distribution of Peak Time Accesses by Size – Statistical Periods 7613 - 7712

Table 5. Analysis of Peak Time Accesses on Edinburgh DECsystem-10 By Size & Date

Size of Accesses	Number of Accesses in Statistical Period													Total
(AUs)	7707	7708	7709	7710	7711	7712	7713	7801	7802	7803	7804	7805	7806	
0-4.99	18	19	21	21	24	24	29	28	28	28	32	31	34	337
5-9.99	5	6	6	3	2	5	3	6	8	9	4	7	3	67
10-	2	4	1	4	3	2	1	2	1	1	2	3	3	29
15-	3	1	1		1	2	3	1		4	2	1		19
20-	2				2			2	3		2	1	2	14
25-				1		1	1	2		1	1	1		8
30-		1		1			1		1					4
35-			1										1	2
40-	1	1												2
45-													1	1
50-			1							1				2
55-					1									1
60-											1			1
65-														
70-				1		1								2
75-														
80-														
85-												1		1
Totals	31	32	31	31	33	35	38	41	41	44	44	45	44	490



EDINBURGH DECsystem-10 INSTALLATION

Total No of Accesses	490
Mean size	5.94 AUs
SE	10.54 AUs

All other details as in Figure 1

To read probabilities, multiply probability scale readings by 0.835.

Figure 2. Distribution of Peak Time Accesses by Size – Statistical Periods 7707 - 7806

5. Applying the Method

Given that the machine has a quantified capcity for the time-band in question, Peak time in this case, the frequency distribution of accesses can be used to set an "upper limit" such that only a predicted number of groups will approach it, and if held at it the machine will not be overloaded. All other accesses can be ignored - in other words only those groups whose demand threatens to overload the machine are controlled and the threat is measured only by the actual usage patterns of the population as a whole and not by any individual characteristics of groups.

The dividing line between an under- and over-loaded machine is not a sharp one, but there must be some measure of capacity if only for general management purposes and the figure used can be refined in the light of experience.

In the Edinburgh Installation the "upper limit" is calculated immediately at the end of each SP, but the frequency required depends of course upon the nature of the installation and the characteristics of its usage.

5.1 Method of Calculation

The method of calculation is given with explanatory notes in Table 6 and is illustrated in Figure 3.

Note that the "upper limit" is set for all projects because it is not known which projcts will reach it - only that 5 are likely to do so, and even if all 5 do, the machine will not be overloaded. Subsequently the"upper limit" is set in the accounting program and the total usage to date in the SP, of each project, at the time of log-in by any member of the group, is checked against it.

To avoid minor and frequent fluctuations in the limit as published and applied, the calculated limit is rounded to 5 AU steps and only changed when shifts greater than this are calculated. Figure 4 shows the movement of the "upper limit" at Edinburgh as the number of research projects on the machine increased and the relationship of the limit to the largest accesses.

In counting the frequencies of accesses, SP by SP, it is important to remember that zero access by a project is a countable event, provided the project has definitely "started" by making some use of the machine in a preceding period. Since projects may or may not continue making accesses until their calendar expiry date, they are not deemed to have "finished" until the expiry date has been reached even though they may be making zero accesses towards the end. For accounting purposes, all allocation periods start at the beginning of an SP and finish at the end of an SP regardless of the actual calendar date when projects actually leave or join the machine.

In the Edinburgh Installation, moving annual figures are used for historical data for forecasting, to remove seasonal effects, but any convenient method of making good forecasts can be used for determining the spare capacity. It is best however, when using statistical method in a practical application, to avoid pursuing accuracy for its own sake. The starting point - the capacity of the machine is a doubtful quantity anyway. Clearly an unnecessarily low "upper limit" results from over-estimating the maximum likely demands and a slight under-estimate is probably better unless a cautious limit is desired. However, some precision is worthwhile in calculating the area under the curve because sharing spare capacity between 6 projects, say, instead of 4 can make a considerable difference in the "upper limit" thus produced.

Although this approach does not use project allocations in its calculations as a means of controlling the usage of resources, and indeed preaches more or less complete freedom for groups to take what is available beneath the upper limit, it may well still be desirable, if not essential, for groups to have some overall allocation or budget imposed for purposes of project or expenditure control, but this would be a measure external to the control of the

Table 6. Calculating the Upper Limit

Representative figures from the year to SP 7806 are used		AUs	Notes
Planned Peak time capacity		662.4	(a)
Deduct maximum likely demand next SP of:			
systems and miscellaneous	69.2		
research projects	254.3		(b)
special reservations of capacity	nil	323.5	(c)
Hence spare Peak time capacity is		338.9	
Large accesses (≥5AUs) will be made by 33% of the groups using the machine 33% of 44 = 14			(d)
Large accesses will consume 86% of the research project load 86% of 254.3 = 218.7			(e)
Hence the average consumption of each large access will be 218.7 / 14 =		15.6	
But some large accesses will rise above the average 12% of 44 = 5			(f)
Hence share the spare capacity between them 323.5 / 5 =		67.8	
Thus the "upper limit" for all groups is or say 85 AUs		83.4	

Notes:

(a) The standard capacity of the machine can be adjusted for known events, eg it would be reduced by 50% if it were known that it was going to be down for 2 weeks in the next SP.

(b) These figures are determined from past actuals, using any suitable forecasting method eg, moving means; Holt's method etc, with suitable adjustments for projects finishing/starting; seasonal variations; public holidays or, for example unusual systems activity.

(c) If a group is known to require a slice of capacity larger than the "upper limit" which must be satisfied, it should be reserved here and the group not checked against the "upper limit" at log-ins during the SP. Special reservations reduce the "upper limit" for other groups and should be made carefully or spare capacity may be held back needlessly if the reservation is not fully used.

(d) The percentage is given by the area under the pdf curve from 5 AUs to the right. The number of projects is that known to be on the machine next SP. Usually new projects starting up, even though known about, can be ignored as being almost certain to fall into the < 5 AUs class. We cannot forecast which 14 projects will make accesses ≥ 5 AUs - it is sufficient to know that 14 (in this example) are likely to do so.

(e) This percentage is calculated from observation of past activity (moving annual or shorter term totals of research group activity) and how much of this is used in accesses ≥ 5 AUs.

(cont on next page)

Installation itself. In Edinburgh, groups may only compute at all provided that they have some unused project allocation. As with the previous rationing system, projects barred by the "upper limit" in Peak time can continue in Standard or Discount time, with the foregoing proviso.

6. Advantages and Limitations

The advantages of the method as against individual rationing are:

(1) It is easier to forecast the behaviour of the population as a whole, than that of its individual elements.

(2) Its nature is to make spare capacity available in full, at the time, to all projects on the machine (or embraced by the control system) leaving them only to the constraints of contention for job slots and what response time is tolerable by them.

(3) It allows machine capacity to be managed as a whole and avoids the need to tinker with individual rations.

(4) It is flexible and can cater for a sudden increase or decrease in the number of projects on a machine, or in the AUs available on the machine.

(5) It provides the information from which management can consider exceptions. For example: if, later in the SP, a group reaches the "upper limit", and there has been a shortfall in the predicted number of large accesses, the management could view a request for more resources sympathetically. Of course, earlier in the SP the management would have to assume that the predicted number of groups will reach the "upper limit" and the request would have to be refused.

It has the limitations that:

(1) It uses a notional capacity, but this is common to most management systems, and it can be refined with experience.

(2) It makes probability assumptions about the number of groups likely to make accesses of particular sizes - but according to the characteristics of the user population these probabilities may indeed be highly predictable.

(3) It uses historical data for forecasting - but then all systems must. The only problem is to get an adequacy of data without it becoming too sluggish or out of date. All calculations must be tempered by local management knowledge.

7. Applicability to Other Installations

The usage analysis described here has been applied retrospectively to the final year of the Edinburgh Installation in its smaller KA10 configuration, when there were fewer groups on the machine and statistics

---

(cont from previous page)

(f) Again this is determined by the area under the pdf curve to the right of the calculated mean size of large accesses (those $\geqslant$ 5 AUs). Using the curve gives a better forecast of the number of groups likely to make "greater than" accesses due to varying raggedness in successive histograms. Also, since the mean size of large accesses is usually not a whole number, the probability can only be calculated or measured from an actual curve. Since here and in (d) above, multiplying the probability of accesses of $\geqslant$ 5 AUs, or $\geqslant$ 15.6 AUs, by the number of projects on the machine again seldom produces a whole number, and since the numbers in this example are small, it is better to round down if the "upper limit" is to be kept as high as possible. If cautious "upper limits" are desired it is better to allow for this by a slight over-estimate of the maximum likely demand.

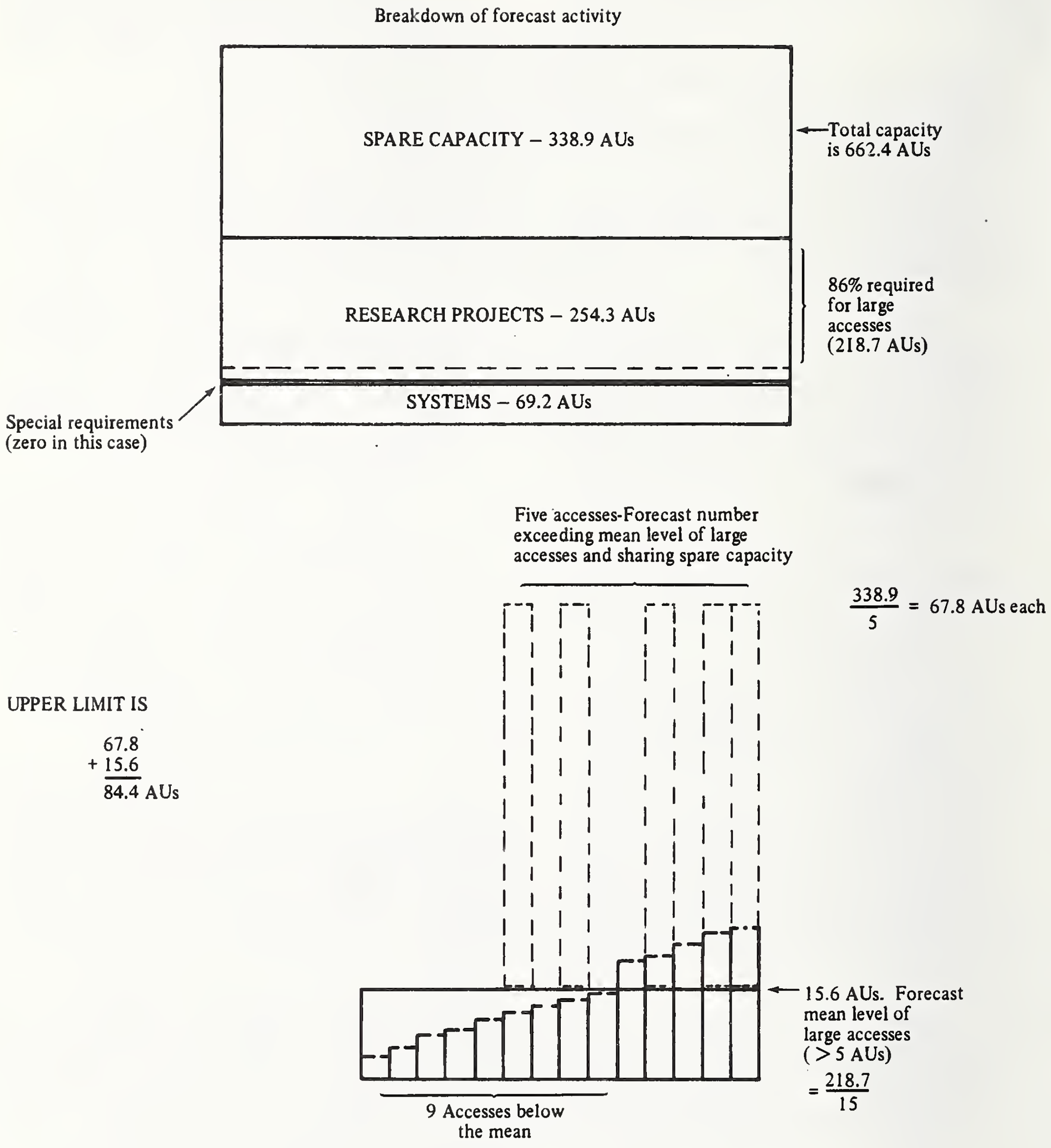

Breakdown of forecast activity
SPARE CAPACITY – 338.9 AUs
Total capacity is 662.4 AUs
RESEARCH PROJECTS – 254.3 AUs
86% required for large accesses (218.7 AUs)
SYSTEMS – 69.2 AUs
Special requirements (zero in this case)
Five accesses-Forecast number exceeding mean level of large accesses and sharing spare capacity
338.9 / 5 = 67.8 AUs each
UPPER LIMIT IS
67.8
+ 15.6
84.4 AUs
15.6 AUs. Forecast mean level of large accesses (> 5 AUs)
= 218.7 / 15
9 Accesses below the mean

Figure 3. Calculating the Upper Limit

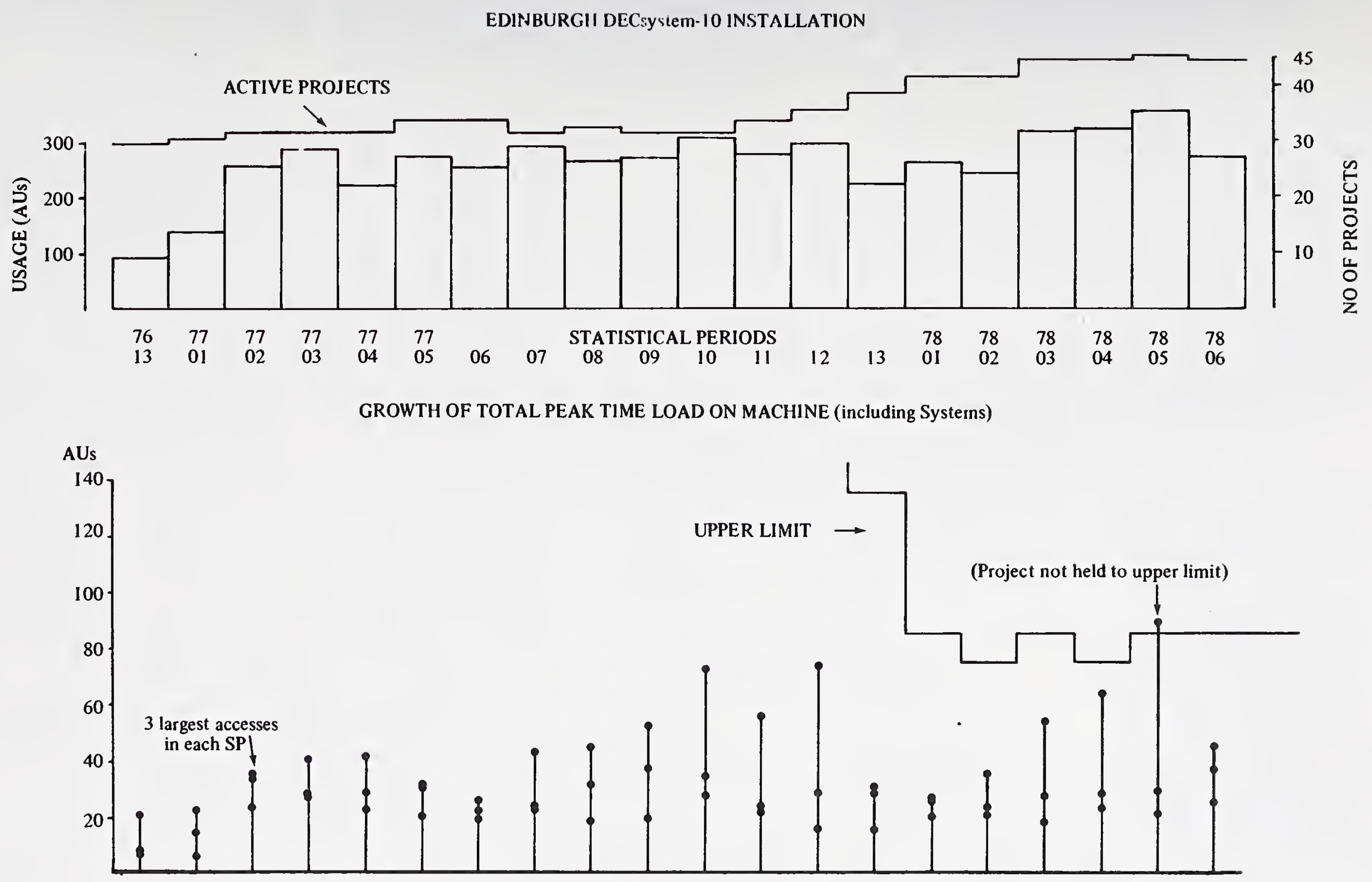


Figure 4. Movement of Upper Limit with Growth of Peak Time Usage & No of Active Projects

were totalled by groups, and not by separate projects. For comparison, the resource usages were converted to AUs by the same algorithm. The combined outcome being fewer accesses of a larger size. The usage figures of a recent year on the Manchester DECsystem-10 Installation of the ICF network have been reviewed also. There they have a slightly larger number of groups making accesses of about half the size of those in Edinburgh. In both cases the same distribution of size of access appeared and quite usable results were obtained.

It therefore seems likely that any time-sharing installation providing general services for a significant number of users, organised in groups or not, will find that the characteristics of user activity can be described by a gamma pdf, whether the activity is measured in one or more computing resources. The larger the user community, and hence the larger the machine, the more predictable the usage pattern is likely to be - but also the larger the potential for waste of spare capacity if control is based on setting maxima for individuals or groups without trying to forecast the actual usage pattern and relating this to the size of the machine.

This is not to say that there is no place for setting maxima at all. In, say, a student population carrying out exercises, the computing content of which is fairly well known, individual control limits, or rations, may well be necessary to protect a teaching department's allocation or budget from waste through ignorance, error, or irresponsibility. Such rationing should be a protection applied at the request of the department, and is harmless if the department can use the saved allocation in some other way. But where rations exceed 10% of nominal capacity there is a danger of resources being wasted through being held back and lost in time.

In an installation with a large batch content it may be necessary to extract batch work patterns in order to detect the time-sharing characteristics, and it may be necessary to have separate control systems.

Annexure A

The Rationing System

Figures A1 and A2 show the usage profiles of 18 projects selected for comparability. They are arranged from top down (from A1 to A2) in ascending order of size of allocation,and hence of minimum resource quota. Projects 1-5 have the same size of allocation, as do, separately, 10 & 11; 12 & 13 and 15 & 16. Forecasting such profiles over a 3 year period for each project separately is clearly impracticable - but it is of course the sum of each of these in each SP which is the theoretical load on the machine.

Figure A3 illustrates the rationing system. Most projects at Edinburgh were running with an of 0.3 and of 1.3.

Figure A4 shows the mean, max and min access of the same 18 projects over a year from SP 7613- SP 7712 (15 Dec 76 - 13 Dec 77) in relation to their Minimum Resource Quota. Their maximum Resource Quota would include the carry forward element which varies for each project from SP to SP and can only be represented as on Project 15. Even without their 'carry forwards' most projects were working comfortably within their ration. The rations are derived from the total allocation and it is difficult to pitch these just right. Many projects need, ideally, a profiled allocation so that their Resource Quota is not just a flat rate throughout the life of the project - but see Figures A1 and A2.

Figure A5 shows the usage in Peak Time of each of these 18 projects in one particular SP (7710). The sum of their minimum resource quotas was 463 AUs (and remember that this is ignoring their carry forwards) but their total usage was 163 AUs. Of course they will have made use of the machine in Standard and Discount time - but they are not obliged to do so. If, therefore, each group's minimum resource quota was regarded as a reservation of capacity and if, for a moment, the machine is considered to have been fully allocated or loaded at 463 AUs, it was only 35% utilised in this SP - 65% of the machine's capacity was wasted. Clearly allocations, or their derivatives, (SP rations) cannot be used as predictors of usage or measures of loading - because actual usage and resource quotas are independent variables.

The large project in the 2nd column of Figure A5 appears to have reached its limit (in fact its carry forward saw it through) - but had it asked for more resources to keep it going the answer, theoretically, would have had to be no, because it could not be known until the SP had finished how many, or few, projects were going to consume all their ration. Obviously, a machine can be allocated on the assumption (gamble) that not all the projects will take their ration. But there is nothing in the total of the allocations to say when the over-allocating has to stop; this can only be decided by observation of actual usage.

Projects are arranged, from top, in ascending size of minimum resource quota, to common scale of AUs.
Projects 1 - 5 have equal quotas (5.01 AUs); 10 & 11 have 20 AUs; 12 & 13 (Figure A2) have 25 AUs.

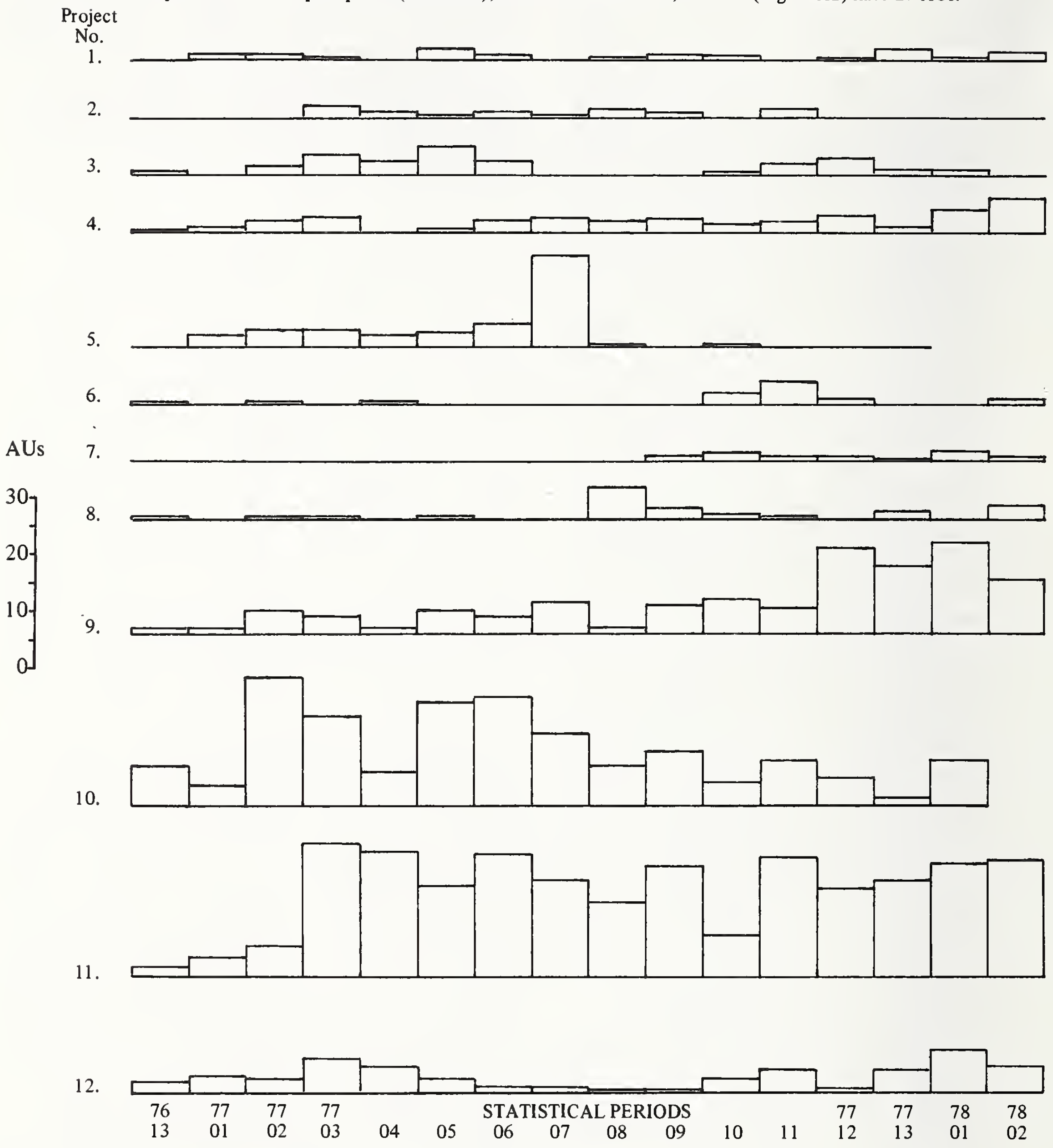


Figure A1. Typical Peak Time Usage Profiles (AUs) – 18 Research Projects on Edinburgh DECsystem-10 (cont in Figure A2)

See Figure A1 for explanatory detail. Projects 13 & 12 (Figure A1) have same allocation (25 AUs), as do Projects 15 & 16 (50 AUs). Project 17 has 70 AUs, and Project 18 90 AUs.

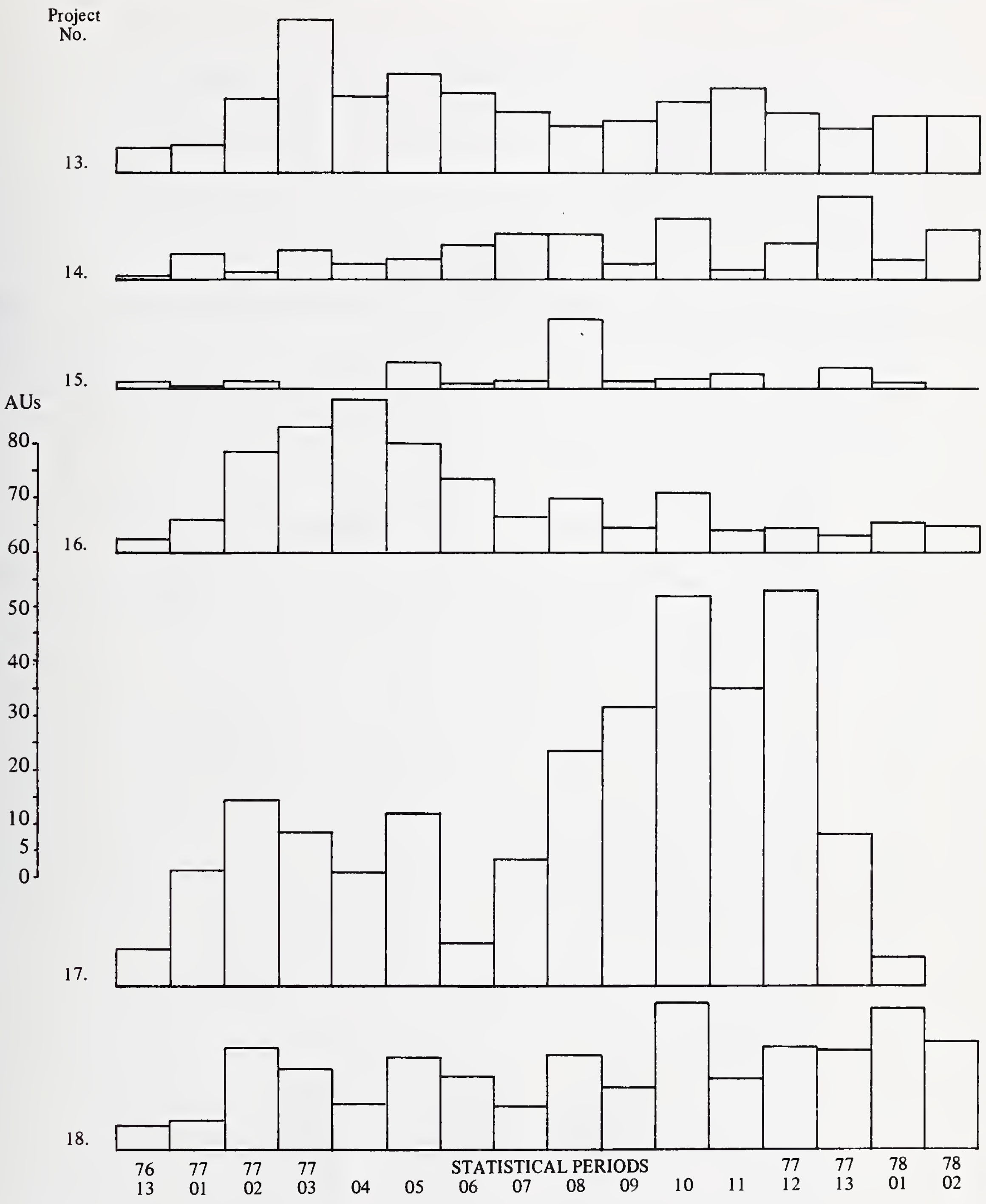


Figure A2. Typical Peak Time Usage Profiles (AUs) – 18 Research Projects on Edinburgh DECsystem-10 (cont from Figure A1)

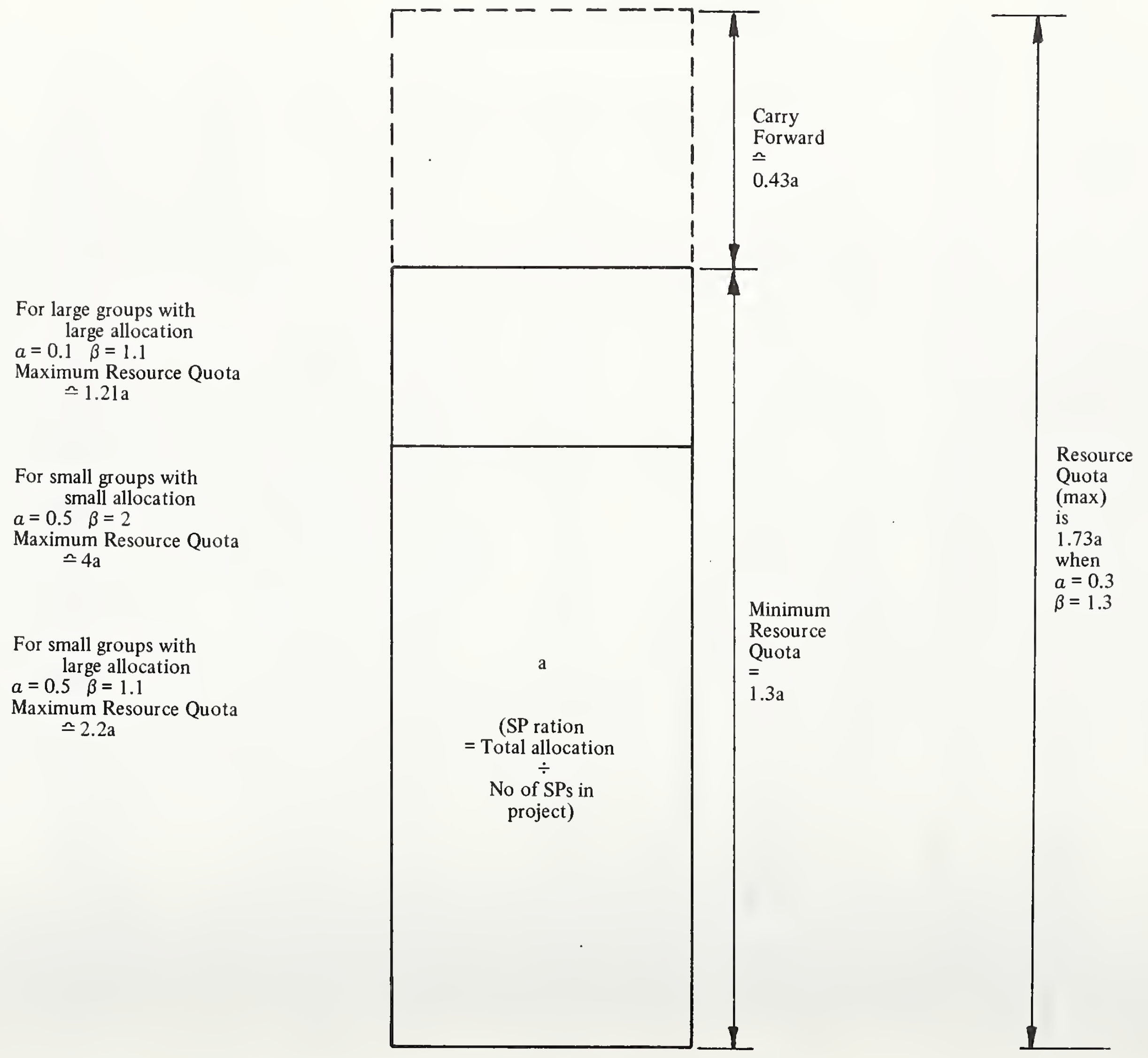

Carry Forward ≏ 0.43a
Resource Quota (max) is 1.73a when $a = 0.3$ $\beta = 1.3$
For large groups with large allocation $a = 0.1$ $\beta = 1.1$ Maximum Resource Quota ≏ 1.21a
For small groups with small allocation $a = 0.5$ $\beta = 2$ Maximum Resource Quota ≏ 4a
For small groups with large allocation $a = 0.5$ $\beta = 1.1$ Maximum Resource Quota ≏ 2.2a
a
(SP ration = Total allocation ÷ No of SPs in project)
Minimum Resource Quota = 1.3a

Figure A3. Edinburgh DECsystem-10 Rationing System

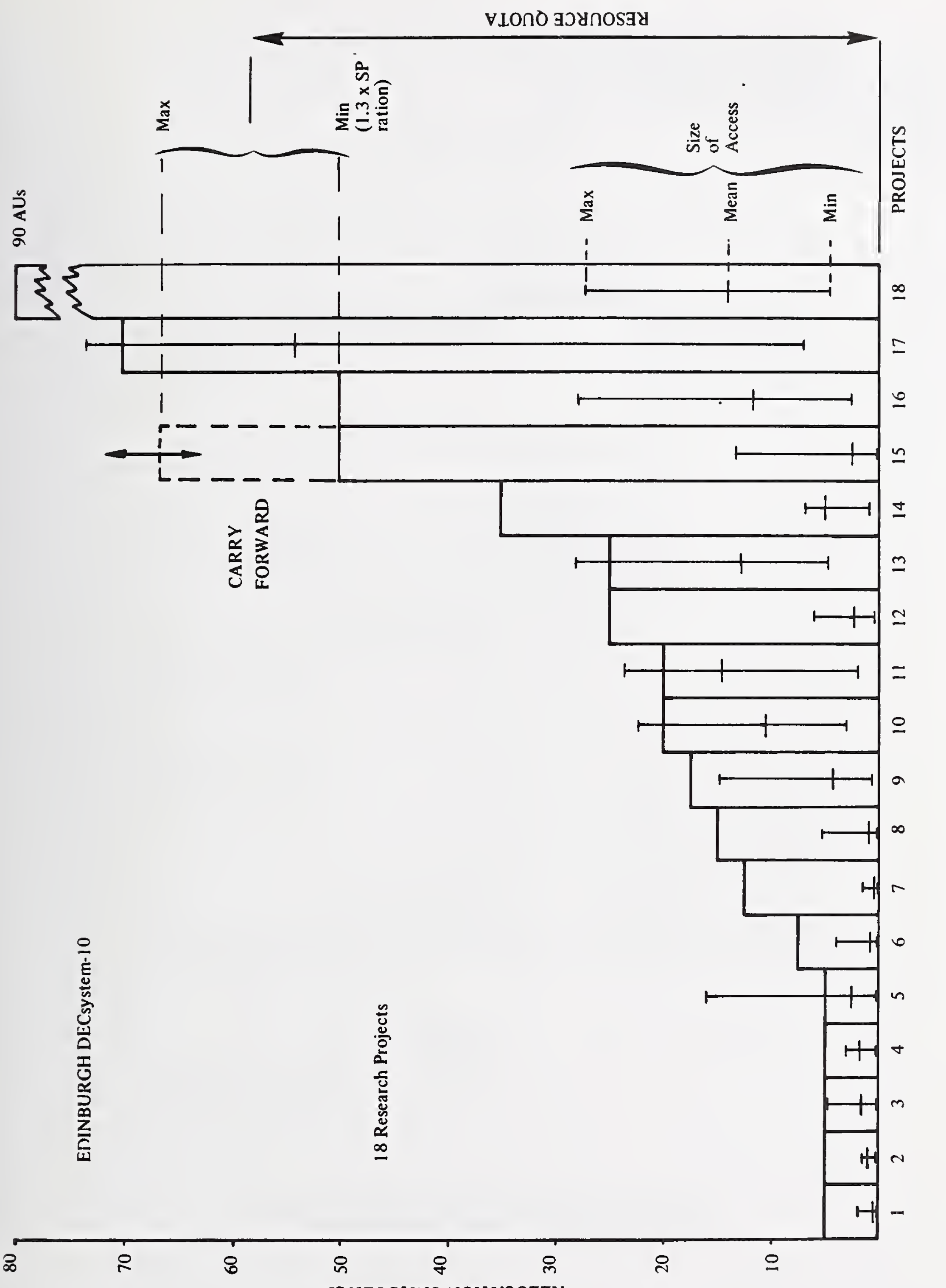


Figure A4. Relationship Between Minimum Resource Quota and Size of Peak Time Accesses In SPs 7613 - 7712

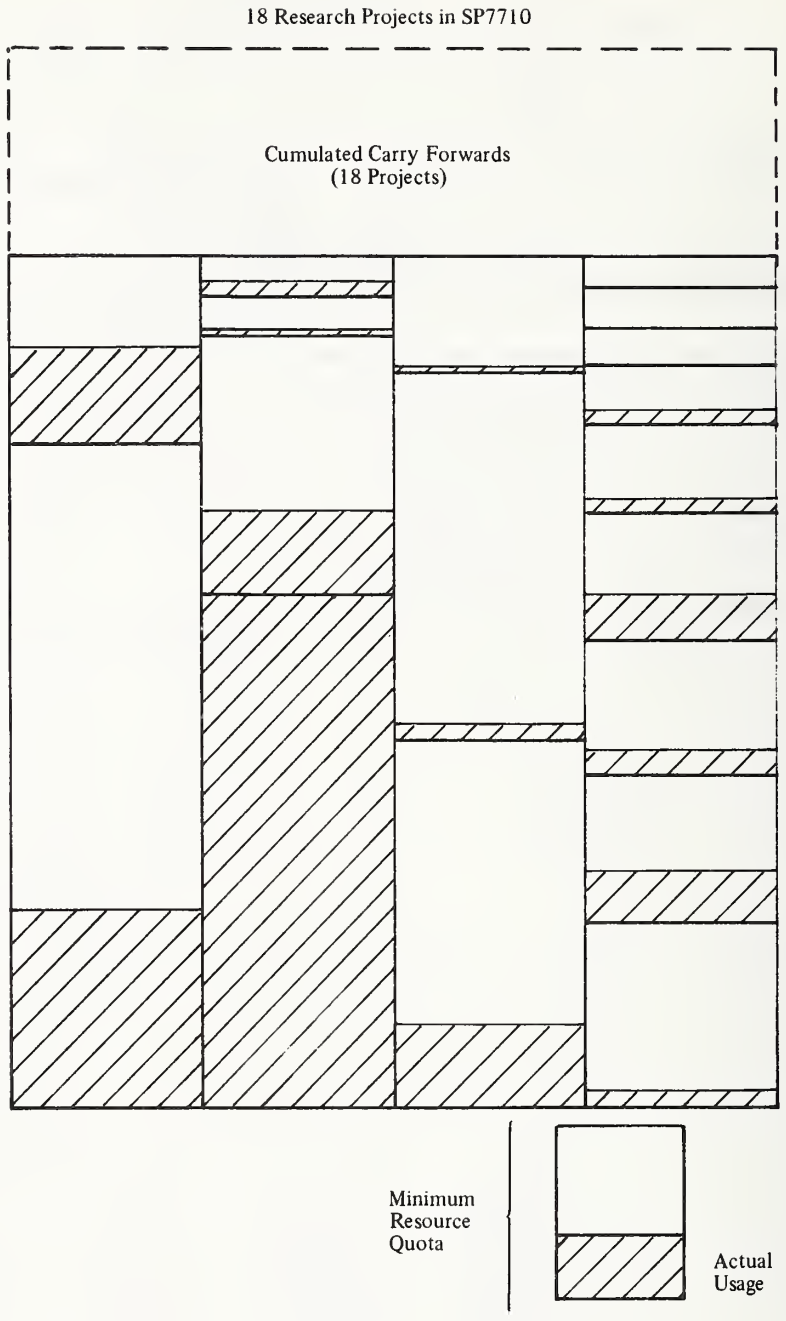


Figure A5. Minimum Resource Quotas (Loading) and Peak Time Usage on Edinburgh DECsystem-10

CPEUG 78

PERFORMANCE IMPROVEMENT PART II: APPLIED STATISTICS

PERFORMANCE IMPROVEMENT PART II: APPLIED STATISTICS

Richard A. Lejk, Colonel, USAF

Deputy Assistant for Logistics Management Systems
Headquarters Air Force Logistics Command
Wright-Patterson Air Force Base, Ohio 45433

This sesson will cover three topics which are different in substance and nature but rely on statistical analysis to support their conclusions. The authors of these papers have provided a sound basis for practical statistical application in measurement evaluation and prediction technology.

In the first paper, "Variability in System Accounting Data," Davies presents the results of a study of the variations in execution times of CPU-bound programs. The author acknowledges two sources of variation -- one which would occur even in a non-multiprogrammed system or environment and the second which is non-repeatable and is based on the system "charging" algorithm. The author points out clearly that these variations must be recognized and considered by the performance analysts when conducting various studies. Lack of recognition results in time wasted obtaining unattainable accuracy. Davis uses simple and compound Poisson's as the basic distributions for the analysis. Data recorded include total elapsed and CPU times, and the mean, standard deviation, and maximum of the CPU times for the individual executions of the test code. The author uses histograms, scatter plots, and regression plots to present the results.

The second paper, "A Statistical Comparison of the System Performance of Several Configurations," describes the study and results of comparing four configurations of a system. Marathe, the author, uses two factors at two different levels to provide the four different configurations. These factors are main memory size (large and small) and the type of disk (Type 1 and Type 2). Through the use of standard t-tests and analysis of variance, the author concludes that all of the factors and their interactions are statistically significant, which leads, in turn, to the use of regression analysis to complete the study. Marathe's use of "dilation" -- the ratio of the elapsed time of a command in a multiprogrammed system to the elapsed time of the same command when only one user is running on the system -- is interesting. This paper likewise uses data plots to present the basic results of the dilation analysis.

Hasche and Grace, in their paper, "Reliability Modeling of Computer Systems," develop a presentation technique so that management is better able to understand computer reliability in order to do effective planning and control. The authors gathered data on ten systems representing four different lines and manufacturers. The data were collected over a 12-month period when no significant changes occurred in the operating environment. Various statistics were computed, such as mean, median, standard deviation, range, skewness, kurtosis, and coefficient of variation for each of the systems. With the test for goodness-of-fit satisfying the assumption of exponential distributions, the probabilities for Times–to–Failure were computed. Using the fifty percentiles, the Mean-Times-to-Failure were computed for each of the ten systems. The authors conclude their paper with the results of validating the model to actual failure history.

In all three papers, the discussions are succinct, the conclusions are encouraging, and the challenges for further studies are issued.

The papers will be presented in the same order as above, with a brief question and answer period to follow. After this general discussion, the presenters will sit as a panel to answer further questions and to discuss the conclusions from the other two papers and their impact on the third paper. Thus, should Marathe be concerned about the variability of timing results validated by Davies? What effect, if any, does MTTF have on Marathe's configuration analysis or Davies analysis of system accounting data? All will be asked to participate in an attempt to resolve these issues and to raise others.

A STATISTICAL COMPARISON OF THE SYSTEM PERFORMANCE OF SEVERAL CONFIGURATIONS

Madhav Marathe

Systems Performance Analysis
Digital Equipment Corporation
Maynard, Mass 01754

This paper describes a study comparing the performance of four configurations of a system. The four configurations were obtained by selecting 2 levels for each of the two factors under study (main memory size and the type of the disk). Standard statistical techniques like the t-test, ANOVA and regression analysis were used to detect and to quantify the differences in the performance of the four configurations. A program development type workload generated using a remote terminal emulator was used for the comparison. The criterion variable was the elapsed time for certain non-trivial commands in the workload.

It was observed that one configuration performs significantly better than the others. The ANOVA procedure showed that all the factors and their interactions are significant in determining the value of the elapsed time. A regression analysis for each combination of the factors was therefore performed. The analysis was used to quantify the additional number of users that can be supported using the better configuration for the same level of user perceived performance. The confidence interval around this value of the additional number of users was also calculated.

Key Words: ANOVA; regression; statistical comparison; system performance; t-test

1.0 Introduction

In the study of computer system performance, many times one faces a task of comparing the overall system performance for a number of different configurations. Such a study is needed to quantify the advantages of a configuration over some other configuration(s) in order to do a cost benefit analysis. Some of the commonly used performance parameters in these systems are the elapsed times for specific benchmarks, the response times for short time-sharing commands, the CPU utilization and `dilation' which is defined as the ratio of the elapsed time of a command in a multiprogramming system to the elapsed time of the same command when only one user is running on the system. The task is complicated in a complex multiprogramming or time-sharing system, since the value of a performance parameter chosen for comparison is not constant, but depends on the interactions of many factors in the system. For example, the elapsed time of a command in a multiprogramming system depends on many uncontrolled factors such as the workload presented by the other users during the execution of this command, the free space in the main memory and the seek and rotational latency times of the moving head disks in the system. A comparison of the two elapsed times has to take into account the inherent randomness in each of the elapsed times.

This paper describes a successful application of standard statistical techniques for comparing elapsed times and for making quantitative statements regarding them. The same techniques can be used for other performance criteria as well. Moreover, even though this paper presents an evaluation using only one kind of workload, statistical techniques exist for comparisons involving many workloads.

2.0 Goals of the Project

1. To determine if there is a statistically significant decrease in the elapsed time for non-trivial commands in a multi-user program development type system when using a Type 2 disk instead of a Type 1 disk.

2. Assuming that the type of the disk, the amount of main memory and the number of users are the only factors affecting the elapsed times, to determine the factors and factor interactions which have a statistically significant effect on the elapsed time for non-trivial commands.

3. To determine, if possible, the additional number of terminals that can be supported for the same level of service by using a Type 2 disk instead of a Type 1 disk.

3.0 Analysis Methodology

3.1 Overall Approach

If an overall model (simulation or analytical) of the behavior of our systems were available in sufficient detail along with a characterization of the desired workload, there will not be any need to perform studies of this type. For example, we will be able to predict the effects of using the two types of disks by suitably altering the disk parameters. Measurements will then be needed only to verify the predictions of the model. In the absence of such a system level model, we are forced to investigate the effects of different configurations by performing actual experiments.

Since the elapsed time for non-trivial commands is the most important criterion for a multi-user system, we decided to directly compare the elapsed times for such commands. If a time-sharing system was under investigation, we would have had to consider the response time for small trivial commands as well. We decided to measure the elapsed time on each of the four configurations obtained by selecting 2 levels each for the main memory size and the disk type. We also decided to use a remote terminal emulator (RTE) system to simulate the activities of the users since that is the only convenient and reproducible way to generate identical load for the two parts of the experiment. Turner and Levy [1] describe a similar application of an RTE.

3.2 Statistical Analysis

Because of the inherent randomness in the value of an elapsed time for a command, we decided to use simple statistical techniques for determining if there is a significant difference in the user perceived performance for the four configurations. Since we found out that there is statistically significant difference between the systems, we decided to determine which factors and factor interactions affect the elapsed times. We also constructed a regression model relating the elapsed time (actually dilation: a parameter derived from the elapsed time) to the number of users on the system. With the help of the regression model, we determined the additional number of users that can be supported for the same user perceived performance level by replacing a Type 1 disk with a Type 2 disk.

4.0 Factors Considered in our Study

It is clear that in a general study of the elapsed times, we have to consider a large number of factors. However, in order to complete the study in a reasonable time, we restricted ourselves to the following 3 factors and levels:

1. The number of users executing a fixed script (1, X, 2X, 3X ... users).

2. The amount of main memory on the system (small, large).

3. The type of disk (Type 1 and Type 2)

Other factors which could have been considered include the operating system, the type of commands, the processor and the type of communication line equipment used.

5.0 Details of the Experiment

5.1 Operation of the SCRIPT system

The SCRIPT system simulates the activities of the users by sending out characters from a script file over the specified number of terminals. For simplicity, we used the same script file for all our terminals. The SCRIPT system staggers the start of activity on successive terminals to reduce the synchronization problems caused by using the same script file for all the terminals.

The SCRIPT system allows specification of user `think time' between successive command lines in the script file. In our experiments, SCRIPT was directed to introduce a delay of 5 to 7 seconds between receiving the termination prompt for one command and sending the next command.

5.2 Measurement of the Elapsed Time

The SCRIPT system maintains time using the line frequency clock of 16.6 millisecond resolution. Each input and output line of text is recorded in a log file along with a time stamp and the number of the terminal initiating or receiving the line. The log file was therefore used to determine the elapsed time for the selected non-trivial commands.

Because the start of the activity on the terminals is staggered, steady state is not reached until all terminals become active. We assumed that the system was in the steady state from the time the last terminal in the particular run became active by sending a "HELLO" command. The contents of the log file until the logging of this "HELLO" command were ignored.

The execution of the selected commands can sometimes terminate due to some hardware error (e.g. parity, odd address). In such cases, the elapsed time was not calculated.

Since it is tedious to scan the log file by hand, a small BASIC program was written to scan the log file and to calculate the elapsed times from it. The elapsed times from all the runs were combined to produce two data files for input to SPSS. The first file was for performing the t-test. It consisted of 865 records where each record contained a pair of elapsed times using the Type 1 and the Type 2 disk along with the identifying information like the command number, the terminal number and the number of users in the system. The second file was for input to the regression analysis program. It consisted of 1730 records where each record identified the level of each of the three factors under study and gave the elapsed time corresponding to this combination.

6.0 The Statistical Analysis Techniques

6.1 The t-test

Given the randomness in the values of the elapsed times, the question becomes: is the overall elapsed time behavior of one system significantly better than another system? There is a statistical test (called the paired t-test) which can be used to analyze the differences between the elapsed times for the same commands on the two systems and to determine if the difference is statistically significant at any specified significance level (see [2] or any other standard Statistical text for details).

In our analysis, t is calculated as follows: Let X1 be the elapsed time for a command in a system with a Type 1 disk and let X2 be the corresponding elapsed time in a system with a Type 2 disk. We are interested in the difference between such pairs X1 and X2.

Let D = X1 - X2.

Assume that D is normally distributed with mean d. This is a safe assumption since a large number of pairs are being compared. If the sample mean of the D's is D1 and the sample variance is (SD1)^2,

t = (D1 - d)/(SD1*SQRT(N))

where N is the number of pairs. There are N-1 degrees of freedom associated with this value of t.

6.1.1 Hypothesis Testing

The null hypothesis and the alternate hypothesis in our case are

H0: "d = 0"

H1: "d > 0"

that is, if the null hypothesis is true, there is no significant difference between the two disks. On the other hand, if the alternate hypothesis is true, the Type 2 disk is significantly better than the Type 1 disk. We selected a significance level of 1 percent which means that we agree to accept a probability of 1 percent of rejecting the null hypothesis when in fact it is true.

Assuming that the null hypothesis is true, we can calculate

t = D1/(SD1*SQRT(N))

and read off from the standard statistical tables the probability of exceeding this value due to experimental errors. If this probability is smaller than the selected significance level of 1 percent, the null hypothesis is rejected and if it is greater than 1 percent, the null hypothesis is not rejected.

6.1.2 Results of the t-test

Our experimental design allows us to perform several t-tests. The overall t-test considers all the pairs of elapsed times. Since the overall t-test was found to be significant (t = 9.82. Probability of exceeding this value < 0.001), we decided to perform a t-test for all the pairs from systems with a small main memory size. This test showed that the value of t is negative implying that a one-tailed test should not be made in this case. The probability of exceeding a t value of 0.54 is greater than 0.5. Evidently, there is no significant difference at the 1 percent level between the elapsed times in the two small main memory systems, one with the Type 1 disk and the other with the Type 2 disk. Table 1 summarizes the results of the t-tests.

6.2 Dilation

Since the elapsed times for the different commands differ widely, they are not suitable for an overall analysis involving all the commands. We therefore used another parameter called `dilation' which is the ratio of an elapsed time to the elapsed time for the same command in a single user system using the Type 1 disk and small main memory. In other words, dilation represents the `stretch factor' for a given configurations (amount of memory, type of disk) and for a given number of users with respect to the basic single user Type 1 disk configuration. The commands fell into three broad groups with respect to their dilation. The groups can be roughly identified as small Fortran compilations, large Fortran compilations and the `other' (assemblies and loads). We therefore decided to perform our analysis separately for each of the three groups. This helped reduce the standard error of estimation in our regression analysis.

6.3 The Analysis of Variance

There is another statistical procedure called the analysis of variance procedure (ANOVA for short) which can also be applied to analyze the elapsed time data. This procedure analyzes the effects of the factors (the type of disk, the number of users and the amount of main memory) to determine which factors and factor combinations (interactions) affect the elapsed time significantly. If the effects of factor interactions are significant, each factor combination has to be considered separately in the subsequent regression study.

We used the following model for the elapsed time of a command in our ANOVA procedure:

```
E'(i,j,k,l)
 = E0 + U(i) + M(j) + D(k)
   + UM(i,j) + MD(j,k) + DU(k,i)
   + UMD(i,j,k) + error(i,j,k,l)
```

Where,

```
i (1=< i =<5) gives the number of
                users (1,X,2X,3X,4X),
j (1=< j =<2) gives the memory size
                ( small and large),
k (1=< k =<2) gives the disk type
                (Type 1 and Type 2),
l (1=< l =<L) gives the observation
                number in case there
                are L repeated
                observations of this
                particular combina-
                tion of (i,j,k)
```

and

```
E'(i,j,k,l) is an observed dilation
        value,
E0   is a constant term,
U(i) is the effect of the i th number
        of users,
M(j) is the effect of the j th memory
        size,
D(k) is the effect of the k th disk
        type,
UM(i,j) is the effect of the combina-
        tion of the i th number of
        users and the j th memory
        size,
MD(j,k) is the effect of the combina-
        tion of the j th memory size
        and the k th disk type,
DU(k,i) is the effect of the combina-
        tion of the k th disk type
        and the i th number of users,
UMD(i,j,k) is the effect (i,j,k) th
        combination of the three,
error(i,j,k,l) is the experimental
        random error in the l th
        observation of the dilation
        corresponding to (i,j,k).
```

The error term is assumed to be normally and independently distributed with zero mean and a variance = s^2.

The ANOVA procedure analyses the data to determine which of these effects are significant. This information is used in the next phase to determine how regression analysis is to be performed on the data. The regression analysis quantifies the effects of the various factors pointed out as significant by the ANOVA procedure.

6.3.1 Results of ANOVA

Table 2 presents the results of the ANOVA procedure for the three groups of commands. Note that the effects of all the factors and their interactions are significant when all the data is considered. However, the effects of the type of the disk and size of the main memory become insignificant if all the data points for the large main memory size/ Type 2 disk case are excluded. This shows that the effects of the disk type are not significant at small memory sizes and the effects of the memory size are not significant when a Type 1 disk is used. The former of these results was also established by the t-test.

Since all the main effects and their interactions were found to be significant, we cannot quantify the effect of any one factor independent of other factors. A separate regression between dilation and the number of users for each combination of the levels of the other two factors is therefore necessary. Another use of an ANOVA test is to help in the design of future experiments by pointing out the sources of large variances so that more effort can be devoted to their study. However, in our case, all our factors are fixed, that is, the levels selected for each factor are not chosen at random from a large selection of levels. Our ANOVA test is therefore not too helpful to us for designing future experiments of this type.

6.4 Regression Analysis

A linear regression model relating the dilation to the number of users was built. Since our experiment involved only two main memory sizes, we cannot build a model relating the dilation to the memory size. Since the ANOVA procedure showed that the effects of factor interactions are significant, it was necessary to develop four regression models corresponding to the four combinations of the two factors (disk type and memory size). Table 3 displays the regression equations and the errors associated with their predictions for these four combinations across the three groups of commands. The value of R-square indicates the closeness of fit of the straight line to the experimentally measured data. Table 3 shows that all but one of the values are close to 1 indicating a very good fit.

The additional number of users that can be supported at the same level of user perceived performance by replacing an RS04 with an RS0X in a 512 Kbytes RSX11M system can be calculated from the regression equations. However, it should be noted that we are assuming that the users perceive the same performance if the average dilation of two systems is the same. The standard error of estimate of Y is used to calculate the standard error around the point on the X-axis corresponding to a given value of an average dilation (Y-axis) as follows:

The regression equation is

    Y = a + b*X

For a given value of Y = Y0, the corresponding value of X is

    X0 = (Y0 - a)/b.

The standard error of this estimate is given by

```
sx =(standard error of estimate of Y)
    *sqrt(1/n + x^2/sum(x^2))/b
where,
    n = number of samples used in
        the regression line,
    x = X0 - mean value of X
    sum(x^2) = sum of squares of x
```

In terms of the values supplied by SPSS,

```
sum(x^2)
  = ((standard error of B)/
    (standard error of estimate))^2
```

It follows that if X01 and X02 are the values on the X-axis for disk types 1 and 2 corresponding to the same value of dilation (Y-axis), and if sx1 and sx2 are their standard errors,

the additional number of users supported by using a type 2 disk instead of a type 1 disk

    =X02 - X01

the standard error of this estimate

= sqrt(sx1^2 + sx2^2)

The values of the additional number of users and the standard error of these estimates are displayed in table 4.

Figures 1-3 display the dilation as a function of the number of users for the three groups of commands. Each graph contains the four combinations of the disk type and memory size. The observed average dilation points are plotted. The standard deviation at each point is indicated by drawing a vertical line between the average dilation +- the standard deviation. The regression equation is used to draw the straight lines. It is immediately obvious that the standard regression assumption of homogeneous variances acrosss the X-axis is not valid. In fact, we discovered that the standard deviation was proportional to the average dilation. In this situation, researchers tend to use a log transformation to equalize the variances or give smaller weights to the points having a large variance. We did not do this for two reasons: first, the relationship between log(dilation) and the number of users was found to be non-linear. Second, because by not transforming we are effectively giving more weight to the variance observed at larger values of the number of users. This is desirable since we are really interested in the behavior of our system when the number of users is large.

7.0 Conclusions and Further Research

The analysis shows that the type 2 disk performs significantly better than the type 1 disk if a large amount of main memory is used. Moreover, it does not show a significant difference for small amounts of main memory between the two disks.

We have also demonstrated the utility of simple statistical techniques for comparisons among different configurations. The techniques applied here can be directly useful to computer installation managers since they can concern themselves only with a small set of workloads specific to their installation. More work is needed to extend the use of statistical models for analyzing the effects of the configurations under study across several workloads. One way of handling the variation across workloads has been described by Marathe [3] in his analysis of the instruction mixes across several application areas and programs.

A statistical model described in this paper is the first step towards a system level cost benefit analysis model. The greatest utility of statistical models lies in the fact that they constitute the scientific basis on which other models can be built. Jain and Potter [4] describe an application of statistical modelling in a cost-benefit study. Other applications of statistical analysis of computer performance can be found in a collection of papers edited by Frieberger [5].

Appreciation is due to Terry Potter for his encouragement and help throughout this study. I also wish to thank Neil Rich for reviewing several drafts of this paper and for suggesting many improvements.

References

[1] Turner R. and Levy H. "Performance Evaluation of IAS on the PDP 11/70" Sigmetrics Symposium Proceeding, Harvard University, 1976.

[2] Snedecor G. and Cochran W. "Statistical Methods" The Iowa State University Press, Ames, Iowa, 1967.

[3] Marathe M. PhD Thesis, Computer Science Dept., Carnegie-Mellon University, 1977.

[4] Jain A and Potter T. "Statistical Modelling of Computer Performance (A Cost-Benefit Approach)" Proceedings of the 12 th meeting of CPEUG, 1976.

[5] Frieberger W. Editor "Statistical Computer Performance Evaluation" Academic Press, New York 1972.

Table 1. The t-tests

	Number of pairs	Mean elapsed time with Type 1 disk (ticks)	Mean elapsed time with Type 2 disk (ticks)	Mean improvement with Type 2 disk (ticks)	Percent improvement with Type 2 disk	t value	Significant at 1 %
1	512	11110	11079	-30	-0.30	-0.54	no
2	865	8747	10576	1828	18	9.82	yes

```
Legend:
      t-test 1:   Main memory   = small
                    of users    = X, 2X, 3X, 4X
                    of commands = 18

      t-test 2:   Main memory   = small and large
                    of users    = X, 2X, 3X, 4X
                    of commands = 18
```

Table 2. ANOVA Output

Source of variation	Degrees of freedom	All data		"Large main memory/Type 2 disk" exclued	
		F	Significant at 1%	F	Significant at 1%
Command Group 1					
users	4	373.4	yes	347.3	yes
memory	1	67.6	yes	3.7	no
disks	1	32.9	yes	0.0	no
users, memory	4	8.9	yes	0.8	no
memory, disks	1	39.4	yes	-	-
disks, users	4	12.2	yes	0.8	no
all three	4	10.4	yes	-	-
Command Group 2					
users	4	1286.9	yes	1132.9	yes
memory	1	196.2	yes	0.6	no
disks	1	121.2	yes	1.2	no
users, memory	4	20.8	yes	2.9	no
memory, disks	1	172.2	yes	-	-
disks, users	4	16.9	yes	0.7	no
all three	4	31.3	yes	-	-
Command Group 3					
users	4	1922.4	yes	1592.3	yes
memory	1	45.4	yes	70.1	yes
disks	1	507.4	yes	4.4	no
users, memory	4	34.0	yes	85.1	yes
memory, disks	1	461.7	yes	-	-
disks, users	4	135.2	yes	0.6	no
all three	4	113.5	yes	-	-

Note: "-" denotes values which cannot be calculated because one set of data points are excluded.

Table 3.
Dilation vs Number of users

Command group number	Memory size	Disk type	slope	inter- cept	R square	Standard error of estimation
1	small	1	0.618	-0.786	0.838	1.693
		2	0.620	-0.853	0.843	1.649
	large	1	0.608	-1.127	0.755	2.137
		2	0.287	0.824	0.513	1.777
2	small	1	0.947	-0.619	0.933	1.627
		2	0.994	-0.945	0.901	2.129
	large	1	1.002	-1.169	0.900	1.968
		2	0.545	0.314	0.855	1.557
3	small	1	0.906	-0.964	0.932	1.571
		2	0.868	-0.823	0.892	1.979
	large	1	1.293	-4.113	0.738	4.834
		2	0.487	0.083	0.829	1.479

Table 4.
Additional Users with the Type 2 disk and large memory

Group of commands	Dilation level	# of users with Type 1 disk, large main memory		# of users with Type 2 disk, large main memory		Additional # of users with Type 2 disk	
		#	S.E.	#	S.E.	#	S.E.
1. Short Fortran compile	5	10.07	0.39	14.55	0.33	4.48	0.51
	10	18.3	0.80	31.97	2.20	13.67	2.34
2. Large Fortran compile	5	6.16	0.23	8.59	0.15	2.44	0.27
	10	11.15	0.31	17.77	0.31	6.62	0.44
3. Others	5	7.05	0.30	10.01	0.06	3.05	0.30
	10	10.91	0.53	20.36	0.27	9.45	0.59

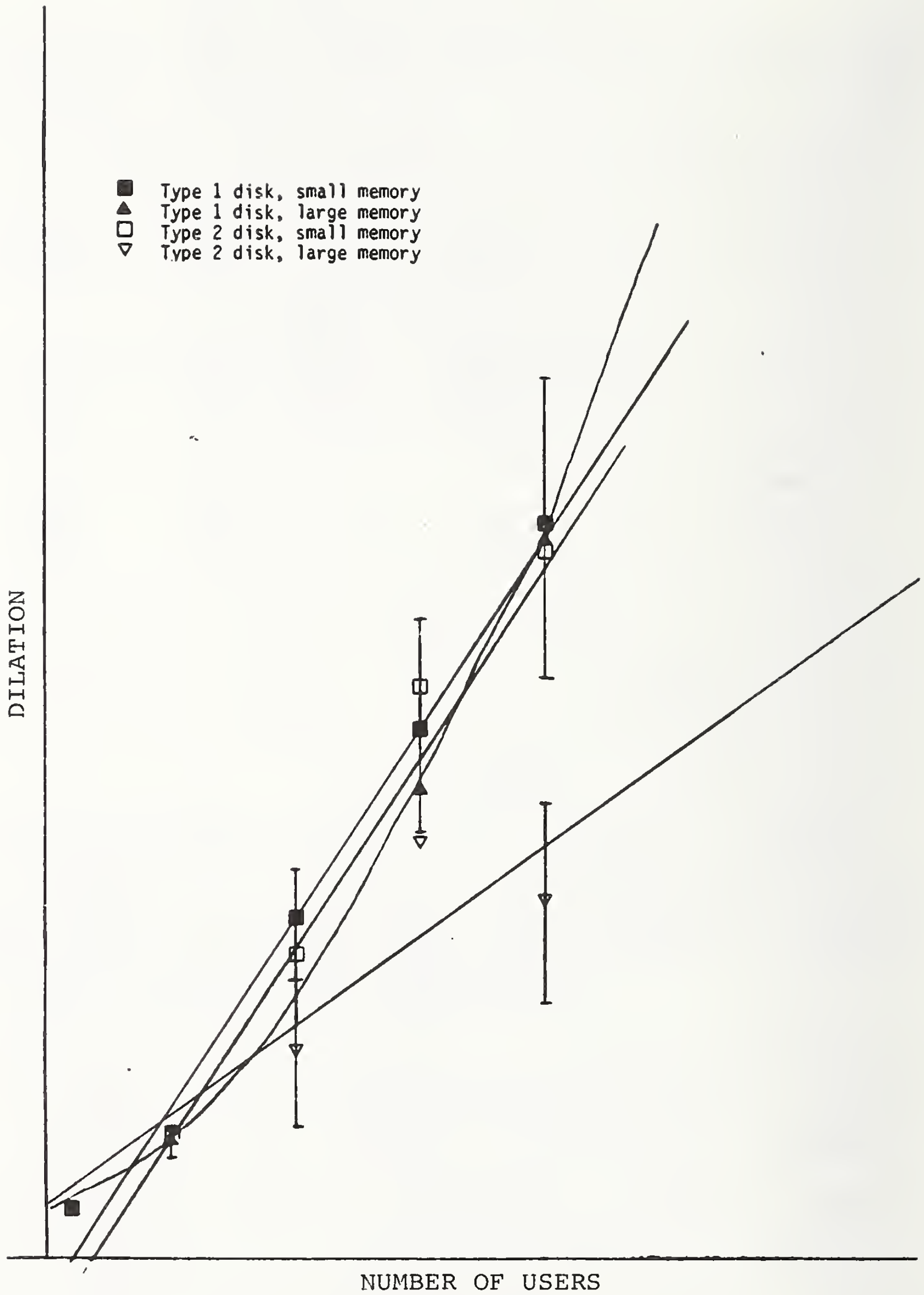


Figure 1. Command Group 1

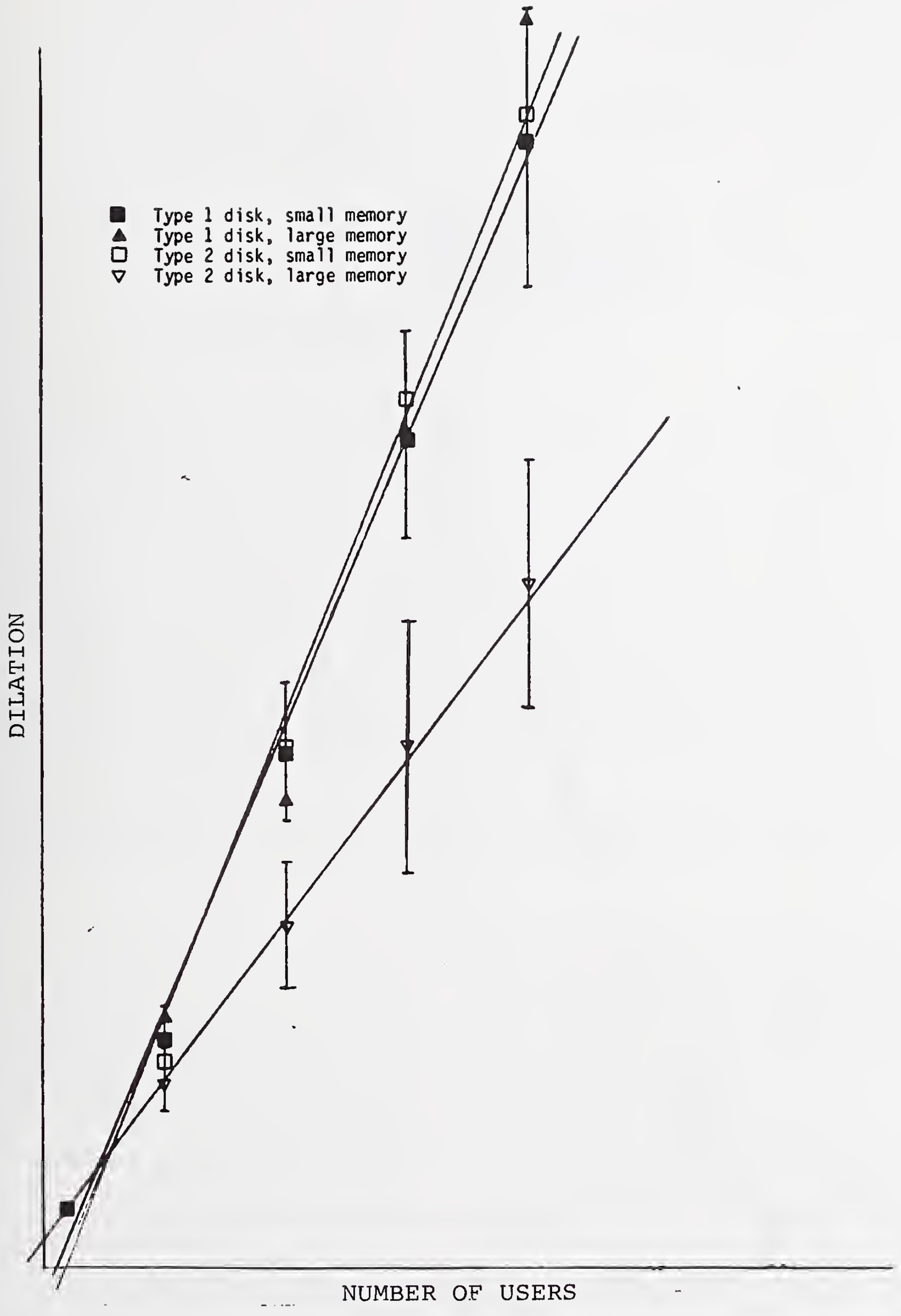


Figure 2. Command Group 2

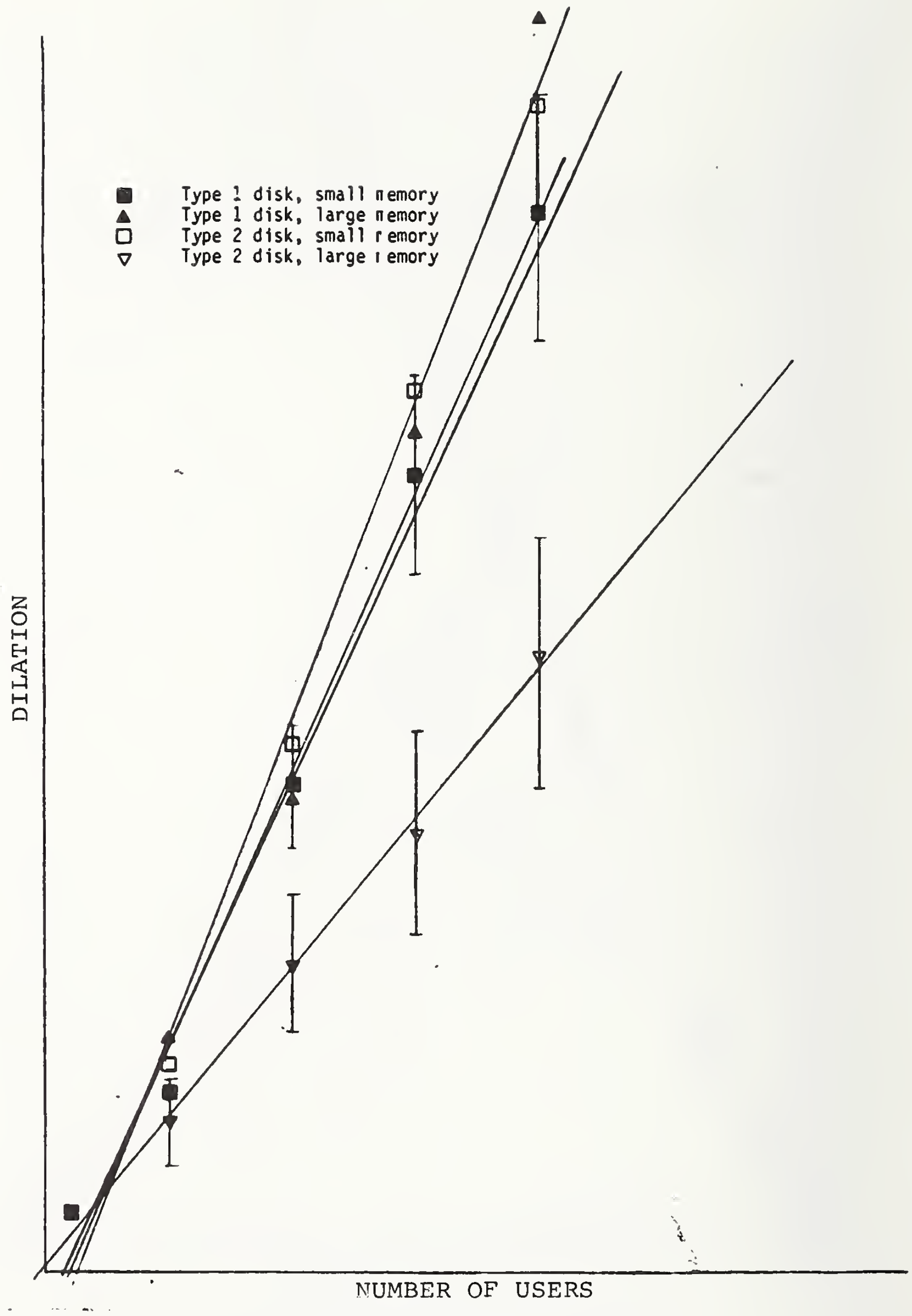


FIGURE 3. Command Group 3

RELIABILITY MODELING OF COMPUTER SYSTEMS

Lloyd R. Hasche
and
Richard A. Grace

Deputy Chief of Staff/Data Systems
Headquarters Strategic Air Command
Offutt Air Force Base, Nebraska 68113

Accomplishment of the Strategic Air Command mission is directly dependent on having reliable computer system support. By having an accurate picture of the expected behavior of each computer system in terms of the probable elapsed time to failure, management can apply the appropriate control to assure a satisfactory level (greater than 98%) of computer system availability. This is accomplished by use of a model that predicts the Mean-Time-To-Failure (MTTF) for a computer system. This paper describes the application of this reliability engineering statistical modeling technique to computer system performance, reviews the mathematical function used, and compares the resulting computer system models to the actual behavior of ten large scale computer systems. The use of this modeling technique has enhanced computer reliability information presentations and contributed to the decisions made on resource allocation to assure computer system reliability.

Key words: Computer system reliability; Mean-Time-To-Failure; reliability engineering; service level management presentations; statistics.

1. Background

Computer System Reliability is a performance factor that management must be able to understand in order to plan and control. A common measure of reliability is needed that is meaningful to the user and sensitive to changes in computer resource allocation. Reliability as a performance characteristic has been developed for electrical systems, mechanical devices, weapon systems and communication systems by measuring the failure-free performance of the total system. Current computer reliability statistics focus on performance of hardware components and software for monitoring of contractual commitments or determining detailed indicators of efficiency. The means to analyze and engineer the reliability of the total computer system has not been fully developed. The Mean-Time-To-Failure (MTTF) statistic and model provide a way to represent current total system performance and predict the impact of resources allocated in improving reliability. In addition, this statistic because of its use in systems engineering can provide the common understanding needed.

The definition of reliability used within the context of this paper is one adapted by the Electronic Industries Association as follows: "Reliability is the probability of a device performing its purpose adequately for the period of time intended under the operating conditions encountered" [1] [1]. It is assumed that a computer system behaves like a device.

[1]Figures in brackets indicate the literature references at the end of this paper.

The failure of the computer system is the unexpected stop in computer service to all users.

The time between failures is that elapsed time from the start of one failure to the start of the next failure.

The Mean-Time-To-Failure is derived from a model of computer system performance. The application of the Mean-Time-To-Failure statistic has enhanced the analysis of reliability performance of ten large scale computer systems at Headquarters Strategic Air Command. This model has been applied to assess the impact of resource allocation decisions to improve reliability and to provide a standard for comparison of current performance.

2. Statistical Technique

The collection of data, analysis of the distribution of data, application of the cumulative probability function and development of supporting graphics used the aid of a desktop minicomputer and supporting statistical programs [2]. No description of these programs is attempted. These supporting capabilities should be readily available. Of interest is the application to computer system Mean-Time-To-Failure (MTTF) statistics.

A history of the time of each failure has been collected for ten computer systems (6 Honeywell 6080s, UNIVAC 1106, IBM 360/85, IBM 370/168 and ADAGE 50 to be referred to as CS1, CS2, ... CS10). An agreement was reached with operations management to determine and report failures based upon the definitions given in paragraph 1. It is assumed that the Time-To-Failure (TTF) is a random variable independent of each occurrence. In addition, length of failure does not affect the rate of failure. The TTF data sample for each computer system was collected over a time period of one year or when no significant changes in operating environment occurred (i.e., changes to hardware, software, operating procedures). The number of failures for each sample ranged from 28 to 181 and contained an average of 69 entries.

The basic statistics were calculated for each data sample. The normal mean, median, variance, standard deviations, range, skewness and kurtosis was determined, (see table 1 for results). These statistics indicated that the data contained a wide range of values, with a median value 42% less than mean value and with a distribution that is apparently skewed to right. The kurtosis also indicated a distribution other than normal.

Table 1. Basic Statistics of Time-To-Failure (Based on a Normal Distribution)

	COMPUTER SYSTEMS (CS)									
	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10
MEAN (HRS)	225	107	47	242	87	321	200	102	99	360
MEDIAN (HRS)	128	67	23	169	70	174	130	24	59	294
STD. DEV	262	137	65	347	88	373	208	179	123	444
RANGE (HRS)	1335	644	360	1767	481	1288	888	847	543	1961
SKEWNESS	2.30	2.01	2.71	2.98	1.68	1.39	1.51	2.53	2.01	2.18
KURTOSIS	9.64	7.34	11.1	13.2	6.8	3.78	4.79	9.2	7.03	7.80
COEFFICIENT OF VARIATION	1.16	1.28	1.39	1.43	1.01	1.16	1.03	1.76	1.25	1.23

A histogram of each data sample was constructed to observe the distribution. A greater frequence of TTF within short intervals was found to be the case for all ten computer systems. A further indicator that the distribution of data was not normally or uniformly distributed was tested by plotting the data against probability plot grids for the normal and uniform distribution [3]. The difference in locus of points supported a rejection of the normal and uniform hypothesis. When the data was plotted against an exponential grid, no rejection of this distribution assumption was indicated.

If the operating environment is kept constant and the system has entered a stable performance state, a constant hazard rate and the exponential distribution can be used to describe the reliability of the system [4]. All of the computer systems had been operational for more than a year since installation; therefore, it was assumed they had entered a state of stable performance. In addition, in complex systems, the exponential distribution is most likely to apply [5]. The analysis continued with the assumption that an exponential distribution could be applied. The exponential probability cumulative distribution function was generated on the TTF data sample for each system. Figure 1 is a representative display of this function.

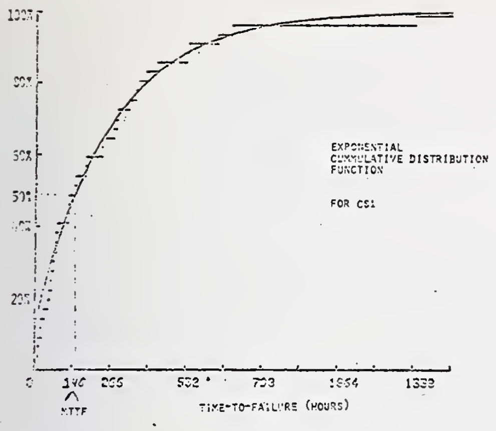


Figure 1. Cumulative Probability as a Function of Time-To-Failure

The Kolomogovov-Smirnov test for goodness of fit indicated that this function was satisfactory to use as a model of reliability for all ten systems [6]. Table 2 displays the TTF for each computer system at the fifty percentile which is the standard Mean-Time-To-Failure (MTTF).

Table 2. Mean-Time-To-Failure Results

MTTF MODEL PERFORMANCE

	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	
TEST 1 50% TTF:	155	74	27	130	74	205	134	47	69	204	(HOURS)
TIME INTERVALS ≤ MTTF	7	10	29	3	10	1	7	3	9	2	
TIME INTERVALS > MTTF	6	10	27	2	12	2	7	6	8	[illegible]	
TEST 2 50% TTF:	146	65	27	141	61	200	136	49	61	226	(HOURS)
TIME INTERVALS ≤ MTTF	6	8	17	2	22	4	4	5	17	0	
TIME INTERVALS > MTTF	4	4	15	5	11	3	5	9	7	2	

The graphic of the model has also proven useful as a practical visual display for management to observe the reliability performance of a computer system.

3. MTTF Model Performance

The actual performance of the computer systems has been compared to the modeled performance of MTTF. The first test compared models based upon six months performance to actual performance over a three month time span. This test was repeated for a model based upon a nine month history. Table 2 displays the results. The results of these two tests established our confidence in the MTTF statistics as being characteristic of overall performance reliability behavior of the computer system.

4. MTTF Application

The MTTF models were used to demonstrate the impact of elimination of types of computer failure (e.g., hardware failure, software failure). The model of a computer system was regenerated after censoring all the time intervals dependent on a type of failure. The marginal improvement in the resulting MTTF is an indicator of the expected improvement (see Figure 2).

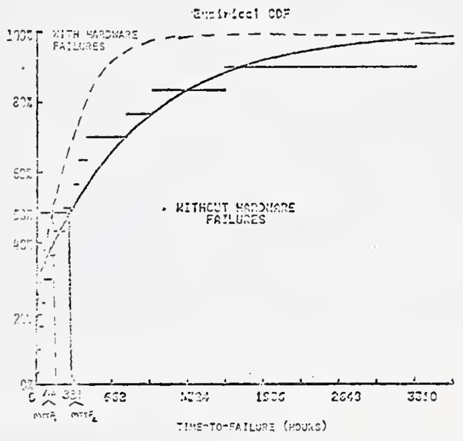


Figure 2. MTTF Model Comparison

A better understanding of the weekly computer system performance results from having established that the distribution of TTF is not a normal or uniform distribution. A suspected trend in decreasing reliability can be confirmed by development of a new model. The resulting MTTF provides a measure of decrease in service. The size of the MTTF can be used to determine the magnitude of the effort to research, analyze, and correct the cause in change in performance.

Finally, when presenting reliability performance to management, the model and use of the term 'Mean-Time-To-Failure' has had wide acceptability. It was readily understood by the general staff during a presentation of the status of computer support.

5. MTTF Future Application

With the MTTF, a means to specify the performance of a total computer system is available. When the replacement or conversions to a new hardware or software system is planned, contractual specification of overall reliability performance can now be delineated. This performance can be demonstrated and documented at the end of a designated performance period.

As computer communication networks become more integrated, the overall reliability planning can be enhanced. The computer component can now be modeled and incorporated into the network model for planning of alternate support paths.

6. Conclusion

With the MTTF model, a means to display and predict the reliability of a computer system is available. MTTF is sensitive to changes in operating environment. MTTF provides a measure of impact of allocation of resources to improve computer system reliability. It also provides a measure of the value of additional information that would be gained by further research into cause and types of computer failures. It has potential for future applications in specifying and modeling computer system performance as a total system or as a component of a computer communication network. Finally, this statistic can contribute to the users' understanding of the reliability service of the supporting computer system.

References

[1] CALABRO, S. R., Reliability Principles and Practices McGraw-Hill Book Co., New York, New York 1962

[2] TEKTRONIX USERS MANUAL, PLOT 50 STATISTICS VOL, 1975

[3] CHORAFAS, D. M., Statistical Process and Reliability Engineering, D. Van Nostrand Co., Princeton, New Jersey 1960

[4] SHOOMAN, M. L., Probabilistic Reliability: An Engineering Approach, McGraw-Hill Book Co. New York, New York 1968

[5] BARLOW, R. E., Mathematical Theory of Reliability, John Wiley & Sons, Inc., New York, New York 1965

[6] WINKLER, R. L., HAYES, W. L., Statistics Probability, Inference and Decision, Holt, Rinehart and Winston, New York, New York 1975

ANALYSIS OF VARIABILITY IN SYSTEM ACCOUNTING DATA

D. Julian M. Davies

Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B9

Results are reported from a study of variations in program execution times, as reported by the operating system of a DECsystem-10/50 time-sharing system. The execution times reported for apparently identical runs of the same (CPU-bound) program vary significantly from run to run. It is shown in particular that the reported CPU consumption rises with increasing system load, and correlations between different measures of program and system performance are analysed.

Two models are proposed for the nature of variations in CPU charges: a simple Poisson, and a compound Poisson model. The empirical results are shown to be reasonably compatible with the simple Poisson model.

This study is of significance particularly to computer users who wish to use program performance metering tools to decide, for example, where effort is justified in 'tuning' a program. Awareness of the nature of variations in execution times permits guidelines to be drawn up to provent a waste of effort by striving for unattainable 'accuracy'.

Key words: Accounting/Computer charges; CPU time, Elapsed time; Metering; System Performance; Tuning Programs; Varying workloads; Variations in CPU charges.

1. Introduction

It is important to be able to measure accurately the time taken by a processor in executing selected sequences of instructions or subroutines. All major operating systems for computers provide facilities for determining how much CPU time is being consumed by a particular process. Such a facility may be based on a continuously running, hardware time-of-day clock, or on an interval timer. The resolutions of such clocks typically vary from 1 microsecond to about 16 milliseconds (a power frequency period).

When a computer is being multi-programmed or time-shared or has a multi-level storage system, the time taken to execute a particular sequence of instructions will vary, even if no I/O activity is involved for the process. The variability, as seen through the operating system timing facility, arises from two different types of cause.

First there is 'true' variation, which would be observed even on a non-multi-programmed system. This arises from variations between runs in cache performance, in paging behaviour in paged systems, in memory access speed if processing is overlapped with "cycle-stealing" I/O, and so on.

The other source of variation in reported time consumption is non-repeatability in how the system 'charges' the elapse of time to the various user processes, to system overheads, and to the idle state. Each user process will have a 'virtual clock' which records the consumption of 'virtual CPU time' by the process. Variations can arise here, particularly in a multi-programming system, for several reasons. There may be delays in stopping and starting 'virtual clocks' at context switches, and the system may not be totally 'honest' in dealing with 'short' interrupts. I/O operations and interrupts will likely vary in CPU requirements from run to run, and systems vary in whether these are charged to system overhead, to the process initiating the I/O operation, or to some other process which just happened to be running at the time ('cheating' at interrupts).

These types of variations have been observed by Bell [1] and Wortman [2]. Wortman was able to compare the magnitude of the "true" hardware variations on a particular task with the variations in times reported by the operating system (OS/360MVT running on an IBM 370/165). It was established that 'hardware time' had a much smaller standard deviation than did the "system time", by three orders of magnitude when timing the same piece of CPU-bound code. Runs of from about 30 μs. to 30 ms. were used, with the clock resolution being about 1 μs.

Bell [1] considered the question of timing variability from the perspective of comparing different systems' charging rates in order to choose the most economical one to compute on. In stand-alone runs of a benchmark, the accounted times showed only small variations on several systems, but with multi-programmed runs there were substantial variations. The variations showed both random and biasing components. The reasons for these variations were as listed earlier: 'biases' or systematic variations particularly arise from 'dishonesty' in time accounting.

Since most users of large-scale systems do not have the privilege of stand-alone access, it is important to appreciate the magnitude and nature of variations in 'system' virtual time consumption. Experimental results and their analysis are reported below for timing information on a DECsystem-10/50.

The management of this particular system have chosen to make the system 'dishonest' in charging for I/O interrupts - they are charged to whichever user process was interrupted rather than to system overhead. This decision, not an uncommon one, is said to have been motivated by a desire not to have the system accounting summaries show a large 'overhead' component. In consequence, however, virtual CPU times reported by the system can vary enormously between runs. Bell mentions a case of one (production) program which when run twice gave CPU charges in the ratio of 1:2. This author has had the same annoying experience.

2. Effects on Program Metering

When conducting timing runs, for example in connection with program or system timing, the effects of these variations must be considered. In many experimental situations where a reading or measurement is subject to experimental errors, the errors can be reduced by taking N repeated measurements and averaging. It is well-known that under fairly general conditions, this procedure will give the mean a standard deviation approximately reduced by $\sqrt{N}$.

However, in the case of computer program metering, this improvement will not necessarily be obtained by repeating the experiment. This is because the experimental data (virtual time reports from the OS) contain systematic errors as well as random variations. The reported times tend to rise as the system load (particularly interrupt load) increases, and the repeated runs may not therefore be taking place in equivalent conditions.

Up to a point, repeated runs of a program will improve the estimate of how much time the program 'intrinsically' uses. Beyond that point, however, further runs will only display variations in the magnitude of the system's load as reflected in the systematic errors. It is of practical and economic importance to identify where that point lies for the particular system.

3. A Model of Timing Variations

This discussion is directed at CPU-bound units of program that have to be timed, but can be extended to accommodate I/O-bound programs. The virtual CPU charge for a program execution can be regarded as the sum of two components. There is the 'physical' CPU time involved in executing the requisite user instructions, and there is system-time - time charged to the user process but stolen at context switches and at interrupts.

The physical time is subject to random variations, as discussed earlier, but these are probably negligibly small compared with 'stolen' time variations in most systems. (The only likely exception is a system with very heavy I/O traffic creating a lot of memory contention, but then one would usually also expect a correspondingly high degree of time 'stolen' at I/O interrupts).

3.1. Simple Poisson Distribution

The 'stolen' time will comprise the 'stealing' of small fragments of CPU time irregularly, but at a rate whose mean increases with system load (degree of multiprogramming, and amount of I/O traffic). The simplest assumption is that CPU time is stolen according to a poisson process, such that a unit of P seconds duration of 'physical' time will be inflated by $\underline{n}$ (stolen) units of length $\underline{s}$, on average every τ seconds.

The measured time M will amount to

$$M = P + ns \qquad (1)$$

where $\underline{n}$ is an integer-valued random variable fitting the Poisson probability distribution [3].

$$p(n, P/\tau) = \exp(-P/\tau)(P/\tau)^n/n! \qquad (2)$$

Both the mean and variance of $\underline{n}$ will be $\kappa = P/\tau$, according to this model. Assuming that the variance in P is negligible compared with the effect of the variance in $\underline{n}$, we have a distribution for the <u>measured</u> time M with

$$\text{mean} = \overline{P}(1\ x+s/\tau) = \overline{P}+s\kappa \qquad (3)$$

$$\sigma^2 = \overline{P}\ s^2/\tau = s^2\kappa \qquad (4)$$

We assume that τ decreases as the system load increases - time is 'stolen' at interrupts, etc. more frequently in a busy system. This leads to increases in both mean and variance of the measured time for program execution.

3.2. Compound Poisson Distribution

The principal approximation made in this model is to assume that all 'stolen' intervals are equally long. In reality they vary, perhaps approximated by another Poisson distribution, which means a Compound Poisson Distribution might be a better fit.

In this case, $\underline{s}$ is also a random variable. If it takes values $0, s_0, 2s_0, 3s_0$, etc. with mean $\overline{s} = \lambda s_0$ and variance λs_0^2, then the distribution for M will have parameters [3]:

$$\text{mean} = \overline{P}+\lambda s_0\kappa \qquad (5)$$

$$\sigma^2 = \lambda s_0^2\kappa(1+\kappa) \qquad (6)$$

This displays similar behaviour, as τ drops, to the simple Poisson case.

4. Experimental Data

A program was run repeatedly on the DECsystem-10, in normal time-sharing under a wide variety of system load conditions. Each run of the program executed a test code 50 times. The test code itself comprised a certain number of iterations of an arithmetic computation: 20, 100, 300 or 1000 repetitions of a computation taking about 1.4 ms. The program recorded the measured vitual CPU time for each repetition of the test code, and also the elapsed and virtual CPU times for all 50 repetitions.

For each set of 50 repetitions of the test code, the program recorded in a data file the total elapsed and CPU times, and the mean, standard deviation and maximum of the CPU times for the individual executions of the test code. These quantities are known respectively as ELAPSE, TCPU, MEAN, STDEV and MAX. In some cases, a histogram was also plotted for the observed times, along with other statistics.

The program structure is shown in Figure 1, and a typical histogram of individual CPU times in Figure 2.

The data from different runs varied according to the length of the test run, and depended on the random variations in system loading. The 'raw' measurements are not independent of each other, and the covariances and correlation coefficients have been analysed. Also, 'scatter graphs' have been plotted, to show the correlations graphically. Figures 4 to 9 show some of these plots. On the plots, the data fall into four groups, corresponding to the four lengths of the test code used.

```
VARS NUMBER#OF#SAMPLES; 50->NUMBER#OF#SAMPLES;


FUNCTION TIMERUN NUMBER#OF#ITERATIONS;
  VARS OLD#CLOCK I J;
  RESETSUMS(); SETSTART();        \for ELAPSE, TCPU
  FORALL I 1 1 NUMBER#OF#SAMPLES
        NUMBER#OF#ITERATIONS->J;
        SYSRUNTIME()->OLD#CLOCK;  \for mean,stdev,max
        UNTIL J=0 THEN J-1->J;
          COPIES#OF#TEXT          \macro inserts copies
        CLOSE;
        ACCUM(SYSRUNTIME()-OLD#CLOCK);
  CLOSE;
  PRDAYTIME();   \output readings to file
  PRRES();
END;
```

Figure 1. Structure of Timing Program

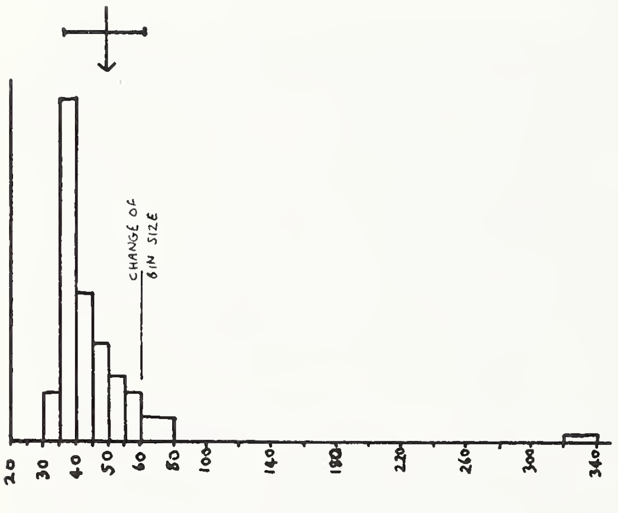


Figure 2. Distribution of CPU times (50 samples)

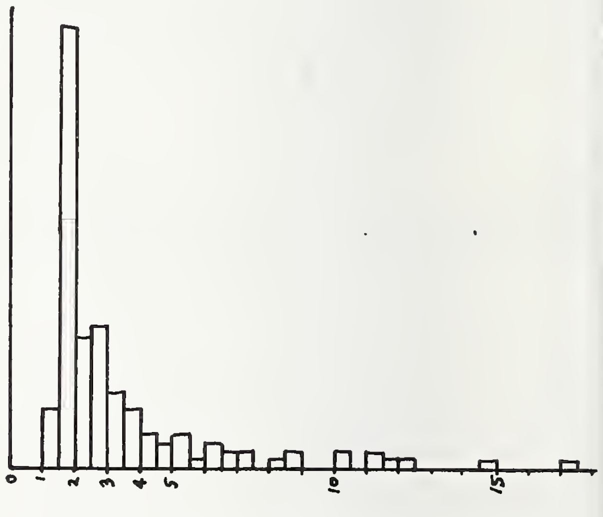


Figure 3. Distribution of STRETCH factors

It was discovered, after collecting some data, that the program was being given inaccurate virtual CPU figures by the operating system. The RUNTIM Monitor call (SVC) was discovered to return the time used as at the start of the current time quantum, not the up-to-date time. The program was then modified to force a brief 'swap-out' just before calling each RUNTIM, to make the system update the accounting meters for the process. The resolution of the RUNTIM clock is 1 ms. The data reported here were collected after that change.

5. Discussion of Scatter Plots

The most direct measure of system load during a run of the program is the total elapsed time for the run. Another measure is the "stretch" factor - ratio of elapsed to CPU time. These are naturally strongly correlated with each other: correlation coefficient 0.98 to 0.99. Table 1 shows all the correlation coefficients. If the CPU times were constant, then the elapsed time and stretch would be exactly proportional. Observed stretch factors ranged from 1.2 to 34 in various runs, with an average of about 3.4 and standard deviation about 3. A typical distribution is shown in Figure 3.

It was hypothesised and observed that the measured CPU times will rise with increasing load. Figure 4 shows the total

Table 1. Correlation Coefficients and Moments

	ELAPSE	TCPU	MEAN	STDEV	MAX		MEAN	Std. Devn
20 Iterations per cycle								
	.984	.772	.740	.465	.361	STRETCH	3.52	3.20
		.806	.778	.507	.393	ELAPSE	10.16 s	12.87 s
			.981	.781	.702	TCPU	2.56 s	0.48 s
				.852	.774	MEAN	39.0 ms	7.3 ms
					.976	STDEV	16.8 ms	16.6 ms
						MAX	134.6 ms	103.5 ms
100 Iterations per cycle								
	.992	.874	.862	.645	.537	STRETCH	3.37	2.45
		.900	.890	.667	.542	ELAPSE	31.63 s	29.45 s
			.996	.821	.727	TCPU	8.60 s	1.26 s
				.846	.750	MEAN	158.3 ms	22.9 ms
					.943	STDEV	32.0 ms	25.2 ms
						MAX	310.6 ms	135.7 ms
300 Iterations per cycle								
	.995	.842	.845	.710	.621	STRETCH	3.33	2.96
		.859	.863	.714	.619	ELAPSE	86.76 s	97.66 s
			.999	.897	.832	TCPU	23.57 s	3.4 s
				.903	.833	MEAN	455.4 ms	65.3 ms
					.956	STDEV	60.2 ms	38.1 ms
						MAX	697.3 ms	189.7 ms
1000 Iterations per cycle								
	.996	.873	.872	.663	.824	STRETCH	3.27	2.55
		.880	.880	.655	.824	ELAPSE	271.15 s	268.91 s
			.99996	.774	.964	TCPU	75.51 s	11.33 s
				.776	.965	MEAN	1494 ms	223 ms
					.850	STDEV	113 ms	74 ms
						MAX	1881 ms	440 ms

STRETCH = ELAPSE/TCPU	MEAN = Mean CPU Time per Cycle	A run
ELAPSE = Overall Elapsed Time	STDEV = Deviation in Time per Cycle	performs
TCPU = Overall CPU Time	MAX = Maximum Time per Cycle	fifty
		cycles

CPU time in a run plotted against the total elapsed time. The best straight-line fits to the sets of data are approximately parallel, which suggests that the increases in CPU and elapsed time from increasing load are proportional to each other over any length of program run. The slope is about 0.05, so the CPU time rises (on average) by 1 second for every 20 seconds of elapsed time. The correlation coefficient is only about 0.8 to 0.9 on these runs. (The CPU times are not quite in the ratios 20:100:300 :1000 because of fixed overheads not subtracted). See also Section 7 below.

The plot of CPU time against stretch factor is shown in Figure 5. The lines are no longer parallel, and the correlation coefficients are all slightly worse, as can be seen from Table 1.

The variations in CPU time (for one cycle) and the maximum individual CPU time both rise with increasing system load. These parameters turned out to be fairly well correlated with each other for the most part, but not so well correlated with the stretch factor or total elapsed time. Figure 6 shows a plot of STDEV against MAX, and Figure 7 is STDEV against MEAN. These show obvious interrelations. Figure 8 shows STDEV against ELAPSE, demonstrating the poor correlation. This lack of correlation is attributed to STDEV and MAX depending on short-term variations in system load, which do not correlate well with the 'smoothed out' loading seen by ELAPSE and STRETCH. The program did not record elapsed times for the individual executions of the test code; it is expected that if these had been collected, they would have correlated strongly with the STDEV and MAX statistics.

Figure 2 shows a histogram of the CPU times for the executions of the test code in a particular run. This histogram is typical, in having most values at the lower end, and a scattering of longer ones. As described earlier, the longer CPU times are attributed to the 'stealing' of time by the system. Figure 7 can be compared with the equations (3) and (4), or (5) and (6), with $\bar{P}$ constant for each set of data and variations in τ or κ (the rate of CPU-stealing) resulting in different data points.

For the simple Poisson case, we can derive

$$\sigma^2 = se \qquad (7)$$

where $e = \text{mean} - \bar{P}$, the excess mean CPU time.

For the compound Poisson case, we can derive

$$\sigma^2 = (e^2 + \bar{s}e)/\lambda \quad . \qquad (8)$$

A plot of E against STDEV^2 (Figure 9) is fitted fairly well by a common straight line, with slope s=95 ms. The random variations from this straight-line fit to the sample data are not small enough to permit discrimination between equations (7) and (8), and (7) is the simpler hypothesis.

6. Analysis of Correlations

Table 1 shows the correlation coefficients and variances of the sample parameters for each of the four run lengths. The correlation coefficients can be analysed by finding the eigenvalues and eigenvectors of their matrix (which is symmetric and positive definite). Table 2(a) shows a typical set of eigenvectors, for the 100 cycle runs. The longest eigenvector is typically 3 to 5 times longer than the next longest, and is contributed to by the original data components in roughly equal amounts. This means that none of the data measurements (stretch, elapse, etc.) is much better than the others for judging system loading, and all vary up and down together, which has already been observed.

The matrices of covariances have been analysed similarly (cf. [4]), and the results are shown in Table 2(b). For this analysis, the ELAPSE and TCPU data were first scaled to ms per run, to make them compatible with MEAN, etc. Most of the variance in the system (the longest eigenvector) is contributed predominantly by ELAPSE, and the next longest is due mainly to MAX. This confirms that the individual variations in the long and short-term timings are largely independent ($\rho = 0.542$).

7. How to Determine 'True' CPU Times

Figure 4 provides a clue to a good way to estimate the 'true' CPU time required by a program. The method is to run the program several times, and record both CPU and Elapsed times. Then calculate the least-squares straight-line fit to the data, and determine where this intersects the line given by CPU=ELAPSE. That will estimate the CPU time that would be required if the computer imposed no extra overheads. Note that the result of this procedure will be an approximate lower bound on the CPU time from a typical run, not the expected time from a run (which will be estimated best by simply averaging the observed times).

Figures 4 and 5 do show some curvature in the upper plots, displaying 'elbows' in the region of stretch=2. Thus a linear relationship between TCPU and ELAPSE is not exactly correct over the whole range of loads. In practical metering, it may be advisable to reject data from runs with a stretch factor greater than 2, on this system, if there are sufficient readings available to permit that.

Corresponding 'elbows' are also visible near the bottom of Figure 9. The significance appears to be that s is smaller on a lightly loaded system than on a busy one, so the simple Poisson model needs some modification.

Figure 4 suggests that if the slope of the CPU vs. ELAPSE line is known, then only one program run would be necessary to form an estimate of the 'base' CPU time. However, since there are sizable variations around the best linear fits, in addition to the elbows, a single reading should not be relied upon for accuracy. It does appear that runs with low STRETCH also show less variance in CPU time.

8. Optimum Length of Run

The data for ELAPSE, TCPU and MEAN for each sample were divided by the number of iterations in the test code, to get the averaged MEAN for 1 execution of the basic arithmetic computation. This normalises the data, except that the means differ because

Table 2. Eigenvectors of Correlations and Covariances

	stretch	elapse	tcpu	mean	stdev	max
1	1.000	0.992	0.874	0.862	0.645	0.537
2	0.992	1.000	0.900	0.890	0.667	0.542
3	0.874	0.900	1.000	0.996	0.821	0.727
4	0.862	0.890	0.996	1.000	0.846	0.750
5	0.645	0.667	0.821	0.846	1.000	0.943
6	0.537	0.542	0.727	0.750	0.943	1.000

EIGEN VALUES

0.003	0.005	0.045	0.141	0.792	5.014

EIGENVECTORS

1	-0.046	0.032	0.681	-0.723	0.096	-0.037
2	0.639	-0.744	0.085	0.028	0.113	-0.129
3	-0.104	0.109	-0.180	-0.026	0.764	-0.600
4	0.469	0.285	-0.547	-0.532	0.125	0.317
5	-0.443	-0.430	-0.090	-0.046	0.468	0.624
6	-0.402	-0.409	-0.435	-0.437	-0.399	-0.363

(a) CORRELATION COEFFICIENTS

	stretch	elapse	tcpu	mean	stdev	max
1	5.852	1392.59	52.578	47.127	38.815	173.640
2	1392.590	337019.01	12994.492	11674.361	9626.969	42100.468
3	52.578	12994.49	619.113	560.177	507.785	2420.638
4	47.127	11674.36	560.177	510.937	475.645	2266.222
5	38.815	9626.97	507.785	475.645	618.058	3134.253
6	173.640	42100.47	2420.638	2266.222	3134.253	17891.034

EIGEN VALUES

0.068	1.509	40.631	125.297	12800.714	343695.77

EIGENVECTORS

1	1.000	-0.005	-0.003	0.018	0.013	-0.003
2	0.014	-0.001	0.650	-0.753	0.098	-0.008
3	-0.011	0.002	-0.268	-0.107	0.949	-0.122
4	0.014	0.041	-0.707	-0.645	-0.257	0.126
5	0.001	0.136	-0.058	-0.060	-0.149	-0.976
6	-0.004	-0.990	-0.039	-0.035	-0.029	-0.129

(b) COVARIANCES

timing overheads have not been subtracted. The means and standard deviations are shown in Table 3.

Table 3. Normalised MEAN data

Number of Iterations	20	100	300	1000
Normalised Mean	1.95	1.58	1.52	1.49 ms.
Standard Deviation	0.37	0.23	0.22	0.22 ms.

This shows that the 'quality' of the averaged CPU-per-executive was improved when timing 100 executions at a time, rather than 20 at a time: a total of 5,000 executions rather than 1,000. However, timing longer sequences of executions than that did not improve the variance further. This residual variance is thought to result from variations in system load. (The same phenomenon appears if the ELAPSE or TCPU figures for the runs are normalised similarly).

By applying the correction described earlier to each of the CPU times (1/20 of excess elapsed time) and renormalising, the variances are reduced but still show the phenomenon, as displayed in Table 4.

Table 4. Normalised Means adjusted for Excess ELAPSE

Number of Iterations	20	100	300	1000
Normalised Adjusted Mean	1.66	1.37	1.31	1.30
Standard Deviation	0.30	0.12	0.16	0.13

On this particular computer, therefore, a metering run should include at least 5 to 10 seconds CPU in the code being timed, but iterations beyond that cannot be expected to improve the final estimate of CPU time for one execution. (If the unit to be tested itself runs for 1 second or more, it should be run at least 3-5 times anyway if a reliable estimate is needed. Recall that even long programs can show up to 2:1 variations in time charged on some systems, though such extremes are rare).

9. Summary

Measurements of program execution time rise with increasing system load. All measures of CPU and elapsed time are positively correlated with each other, and no one of them appears to be a 'best' statistic to use to infer system load. However, longer and shorter term measurements are but poorly correlated in contributing to total system variance.

Changes in observed CPU and Elapsed times for a program are approximately proportional to one another. This forms the basis of a method proposed for estimating the 'true' CPU time requirement of a program being metered. It is observed (on this DECsystem-10) that metering runs should accumulate (optimally) 5 to 10 seconds of CPU time. Shorter runs result in a larger standard deviation in the estimated CPU time, through having too few data. However, longer runs do not subsequently improve the sample variance because of imperfect corrections for variations in system load.

The observed CPU time variations are consistent with a model in which the system 'steals' CPU time from the user job according to a Poisson process, with individual 'stolen' increments of about 95 ms. each. This is a crude model but a good first approximation to the observations. 'Stolen' time is CPU time charged to the user, but used by interrupts and other system overheads. A compound Poisson model was also analysed but not fitted to the data.

It would be interesting to compare these data with results from other operating systems. The results reported in (1) and (2) support similar conclusions with respect to the limits of accuracy obtainable in program metering.

This work was supported in part by an operating research grant from the Natural Research Council of Canada.

References

[1] T.E. Bell, Computer Performance Variability. Proc. Nat. Computer Conf., 1974, pp. 761-766, (AFIPS Press).

[2] D.B. Wortman, A Study of High Resolution Timing. IEEE Trans. Software Engin. SE-2 (June 1976), pp. 135-137.

[3] W. Feller, An Introduction to Probability Theory and its Applications 3rd Edn. Vol. I, John Wiley & Sons, N.Y. (1968).

[4] C. Gonzalez, Using Covariance Analysis as an Aid to Interpret Results of a Performance Measurement. Proc. Int. Symp. Computer Performance Modelling, Measurement and Evaluation, Harvard 1976, pp. 179-186.

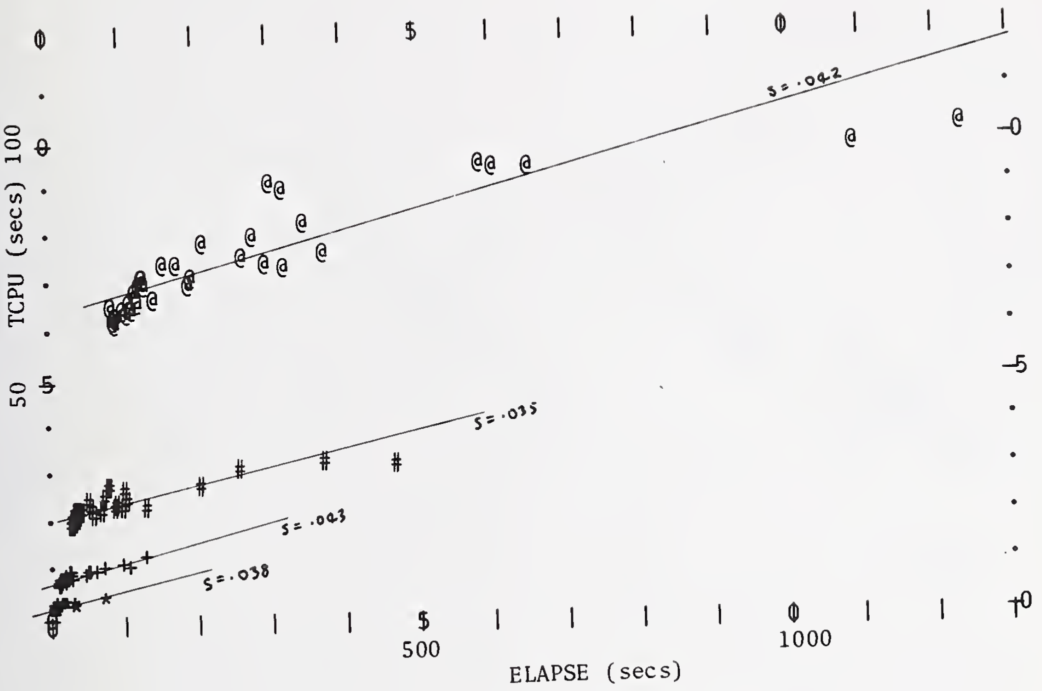


Figure 4. TCPU vs. ELAPSE

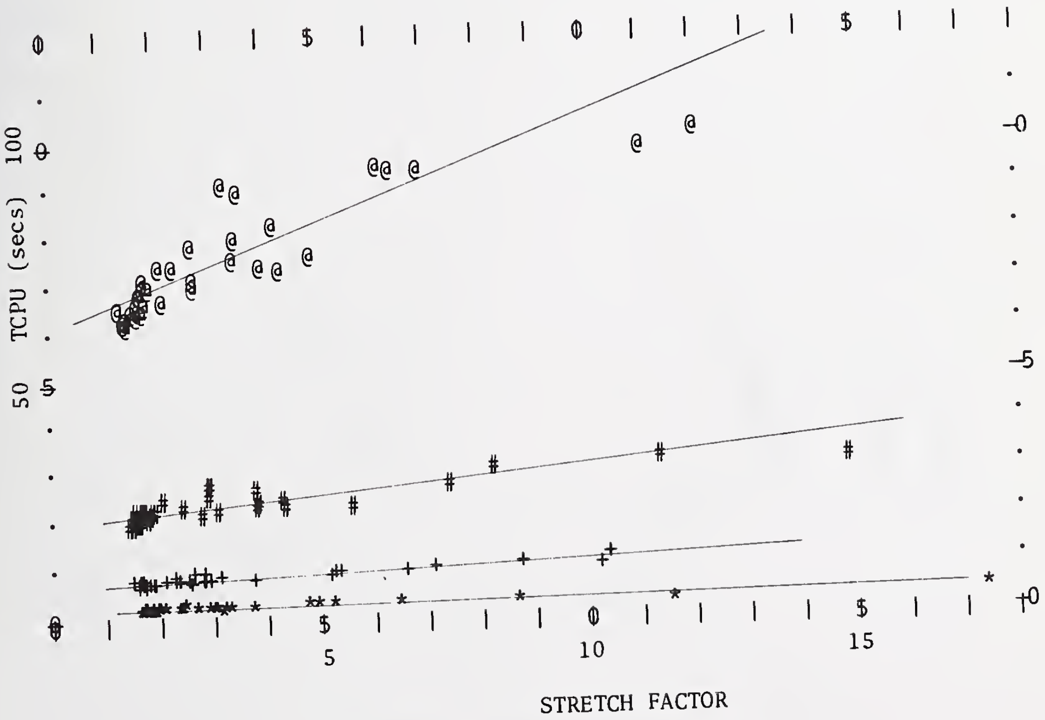


Figure 5. TCPU vs. STRETCH

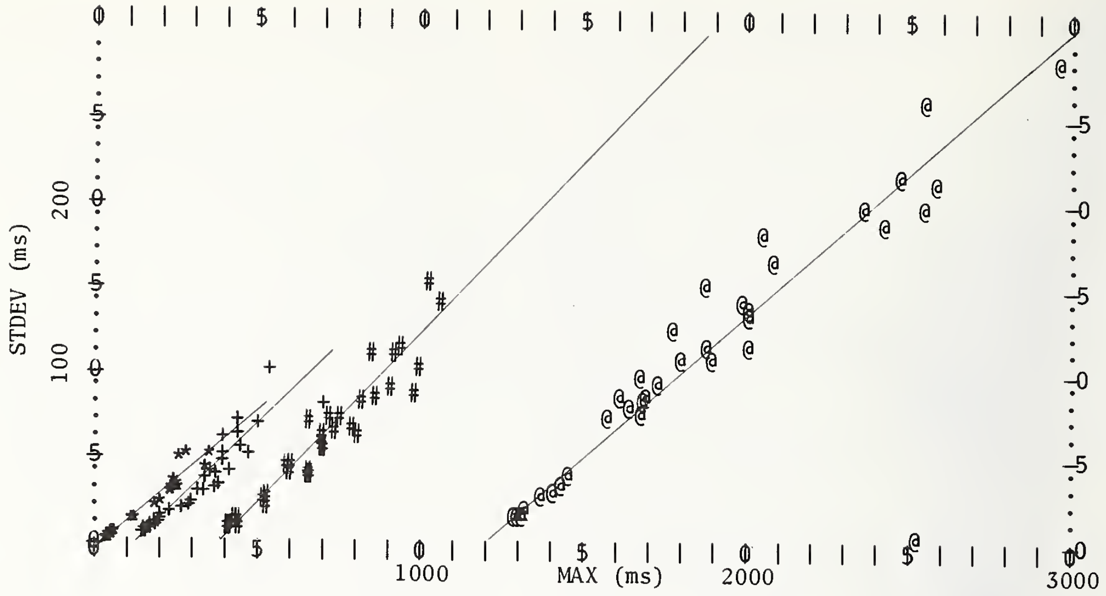


Figure 6. STDEV vs. MAX

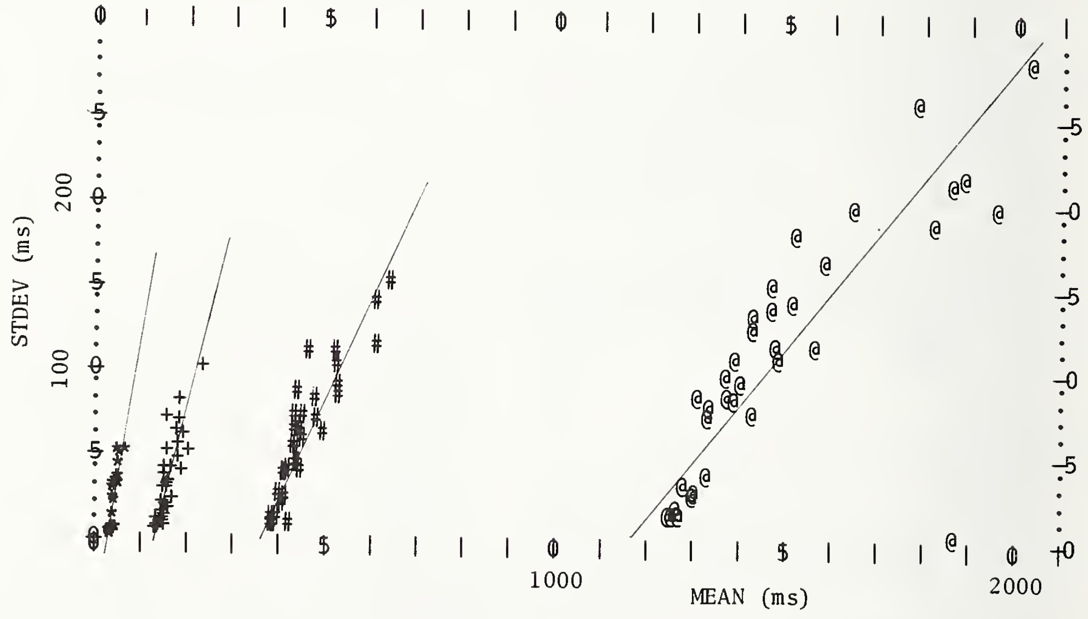


Figure 7. STDEV vs. MEAN

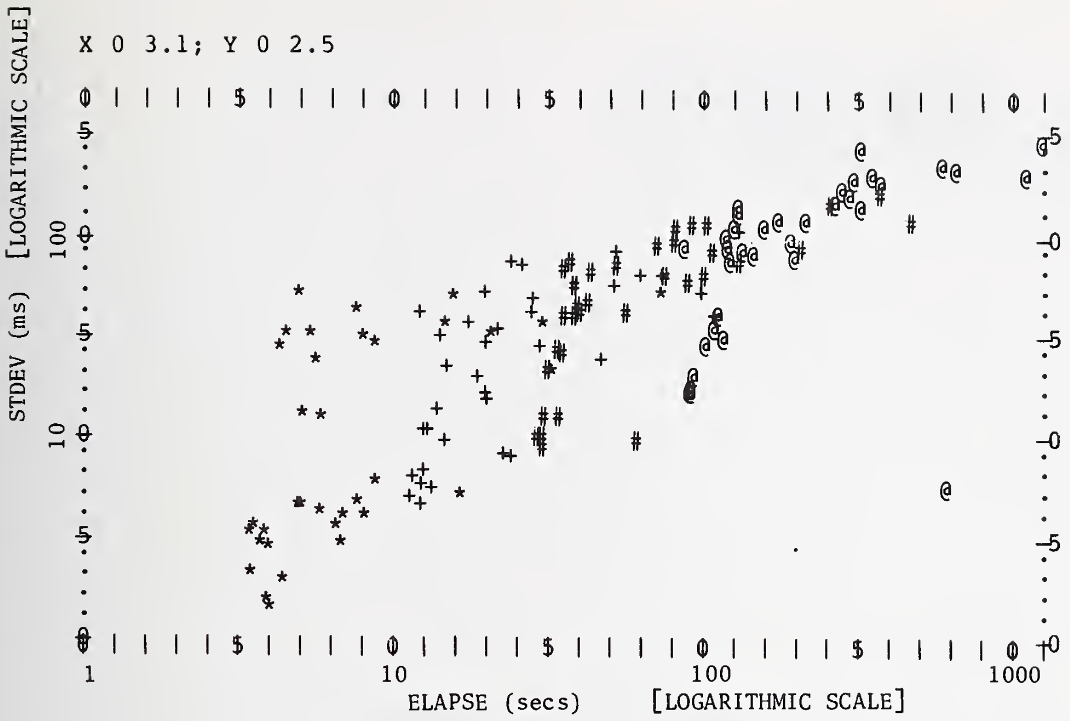


Figure 8. STDEV vs. ELAPSE

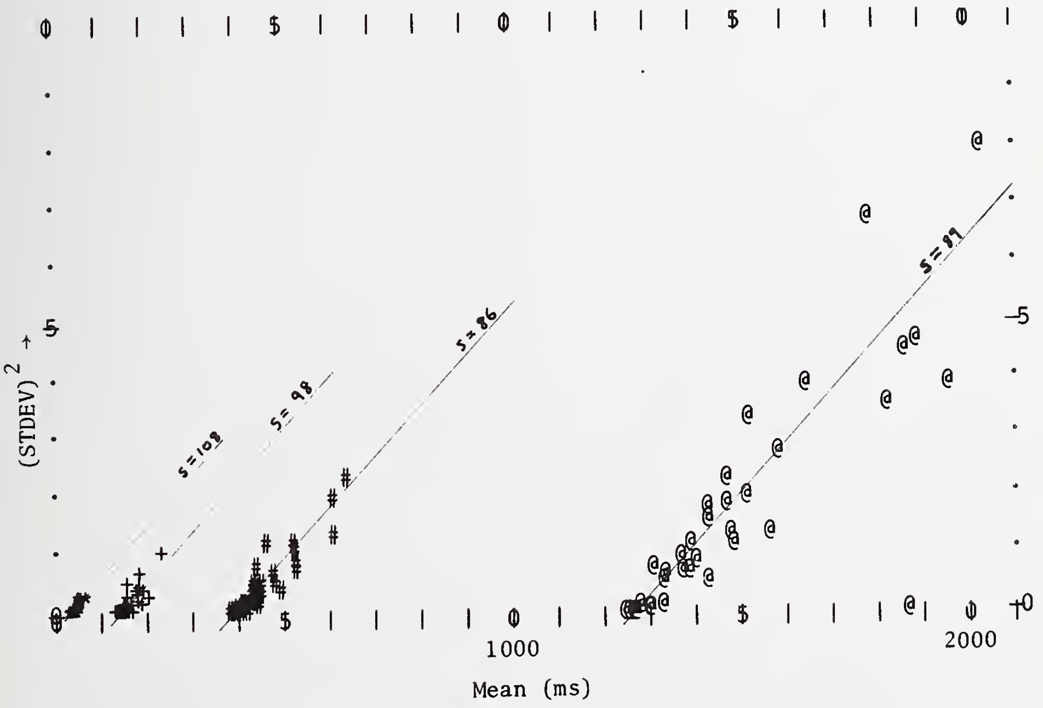


Figure 9. $STDEV^2$ vs. MEAN

CPEUG 78

PERFORMANCE IMPROVEMENT PART III: MEASUREMENT APPLICATIONS

PERFORMANCE IMPROVEMENT MEASUREMENT APPLICATIONS PART III

Phillip C. Howard

Applied Computer Research
Phoenix, Ariz.

This session will cover a wide range of techniques and methods for the measurement and tuning of information systems. The types of systems addressed range from minicomputers to large scale multiprocessors, the techniques from traditional measurement to classical industrial engineering methods. Collectively, the papers illustrate the many alternatives and possibilities that exist for analyzing and tuning computer systems.

Chaikin and Orchard define and analyze a measure of system organization and disorganization, a candidate for a well-defined, single valued indicator of improvement in system performance. The measure is called the System Organization Measure and shares several properties with the Shannon relative entropy function. Potential applications are suggested in the areas of subsystem balance, general system investigation, and system planning.

The paper by Fuller, McGehearty, and Rolf discusses the performance of C.mmp a multiprocessor system consisting of five DEC PDP 11/20's and three PDP 11/40's. In this case, the authors use remote terminal emulation to impose a synthetic load on the system, comparing measured average response times with those predicted by an analytical model. The experiments enabled the authors to refine the model by incorporating additional factors, and to put heavier interactive loads on the system than were experienced under normal situations.

At the University of Regina, a minicomputer system, the DEC PDP 11/34, is used in an undergraduate Computer Science Laboratory. In their paper, Lee, Maguire, and Symes describe a study carried out to measure the performance of this time-shared system. An internally driven measurement package creates a number of pseudo terminal users and an event trace of all activities is recorded. The study indicated how to expand the system from its current thirty to forty or more terminals through software modifications and minor hardware expansion.

The final paper, by Lieberman, employs classical industrial engineering techniques such as availability modeling and the machine interference model to assist in deciding whether or not to configure a multiprocessor from two individual CPU's. The factors considered in the decision framework include the impact on availability, the impact on throughput, and the impact on cost.

A Relative Entropy-Like Measure for System Organization

Jamie A. Chaikin
Robert A. Orchard

Bell Laboratories
Piscataway, New Jersey 08854

ABSTRACT

A measure of system organization and disorganization is defined and analyzed. Its potential role in computer system performance tuning and planning studies is indicated.

In the area of computer system performance and tuning studies, one frequently encounters situations in which component workload processing rates (i.e., component utilizations) must be changed in order to remove bottlenecks or increase the work throughput of the system. In some cases a more equal distribution of workload processing over subsystems of the system is desired, while in other cases a predilection for a particular subsystem may be indicated. It has always been difficult to point to some well-defined single valued indicator of improvement in system performance. The system organization measure (index) introduced in this paper may be a candidate for such a performance improvement index.

1. INTRODUCTION

The analysis and design of complex systems involves, in general, the formulation and solution of problems involving many variables. Comparisons between two distinct settings of system variables is often difficult to carry out. What is desired is some sort of an index or function through which the variable values can be transformed into a single number and the comparison then made.

For example, in the area of computer system performance and tuning studies, one frequently encounters situations in which component workload processing rates (i.e., component utilizations) must be changed in order to remove bottlenecks or increase the work throughput of the system. In some cases a more equal distribution of workload processing over subsystems of the system is desired, while in other cases a predilection for a particular subsystem may be indicated. It has always been difficult to point to some well-defined single valued indicator of improvement in system performance. The system organization measure (index) introduced in this paper may be a candidate for such a performance improvement index.

A typical situation in which this index may be used might be a three component subsystem of a computer system (e.g., channels). If U_1, U_2, U_3 are average channel utilizations observed during one period of time and U_1', U_2', U_3' are average channel utilizations observed during a different period of time, is there any meaningful way of characterizing the relative merits of these two sets of utilizations? As we will see, there is.

The entropy measure developed in this paper is shown to be a potentially useful tool in the study of complex systems from the points of view of both planning and analytic investigation. It also appears that further study in using entropy measures as system wide indicators of performance and service may be fruitful.

2. SYSTEM ORGANIZATION MEASURE

In recent years there have been many attempts to apply the Shannon relative entropy function, [1],

$$H=-\sum_{i=1}^{n} P_i \log P_i / \log n, \quad \sum_i P_i = 1$$

to areas which are, strictly speaking, outside of communication theory. Shannon himself explicitly dissociated himself from those who interpreted his measure of information as a measure of meaning or semantic content.

In an attempt to apply H as a measure of organization and disorganization of a system, the logarithmic nonlinear behavior of H makes it difficult to come up with an interpretation of what a differential increment, ΔH, in H means in terms of a change in the organization of the system. The needed interpretation sought can be expressed from the point of view of workload processing if the entropy measure is generalized.

When a measure is introduced, one is compelled to give a complete quantitative interpretation of its application so that a loose formulation of an initially intuitive concept is not promulgated. With this in mind, a measure (index) was sought which would have similar properties as entropy with respect to measuring deviation from an equally likely or equidistributed system state. In addition it should interpret a differential increment in the index directly in terms of the organization of the system.

Let $S_1, \ldots, S_n$ be n components of a system S and let $u_1', \ldots, u_n'$ be the observed component utilizations over an interval of time Δt.
$u_i' = \dfrac{L}{\Delta t}$ where
L=component i active time in the interval Δt.
The relative utilization, u_i, of component i over the period Δt is defined as

$$u_i = \begin{cases} u_i' / \sum_{i=1}^{n} u_i' & \text{if } u_i \neq 0 \text{ for } some\ i \\ 0 & otherwise \end{cases} \qquad (1)$$

The *balance factor* b, of the system S is defined as

$$b = \begin{cases} \sum_{i \in I} (u_i - f_i) & \text{if } I \neq \phi \\ 0 & otherwise \end{cases} \qquad (2)$$

where f_i is the utilization criteria for balance on component i and I is an indexing set.

In this paper we choose a particular b as follows,

$$f_i = 1/n, \quad \sum_i u_i = 1, \quad \sum_i f_i = 1 \qquad (2a)$$

and

$$I = \{i \mid u_i > \frac{1}{n}\} \qquad (2b)$$

so that

$$b = \sum_{i \in I} (u_i - \frac{1}{n}) \text{ for } n > 1 \text{ and } I \neq \phi$$

$$b = 0 \text{ for } n = 1 \ or\ I = \phi \qquad (2c)$$

b may be interpreted as the fraction of total component utilization $\sum_i u_i'$, which must be redistributed over the components in order to satisfy the utilization criteria for system balance. We note that the assumptions (2a) and (2b) in Eq. (2) have established this criteria as equal "loading" of components. Other criteria may be used.

The *system organization index*, B, is now defined as

$$B(u_1, \ldots, u_n) = \frac{n(1-b)-1}{n-1} \qquad (3)$$

from which we note

$$b = \frac{n-1}{n}(1-B) \qquad (4)$$

Since B is taken with respect to an interval of time Δt, the average organization measure, $\bar{B}$ over total time T is then

$$\bar{B} = \frac{1}{T}\sum_i B_i \Delta t_i, \sum_i \Delta t_i = T \qquad (5)$$

where B_i is the measure in the interval Δt_i.

2.1 Properties of the System Organization Measure

The system organization measure B shares the following properties in common with the Shannon relative entropy function:

1. Both H and B achieve their maximum value of unity when $u_i = 1/n$ for all i.
2. Both H and B achieve their minimum value of zero when $u_i = 1$ for some i and $u_j = 0, j \neq i$.

In addition, if ΔB is an incremental change in the measure of the system organization then its "organizational significance" can be interpreted through Eq. (4), i.e.,

$$\Delta b = \frac{n-1}{n}(1-|\Delta B|)$$

Δb is the fraction of total component utilization (power, work rate) which has been redistributed over the components of the system. If ΔB is positive, this will tend towards uniform loading of the components (disorganization or system balance). If ΔB is negative this will tend towards selective or biased loading of the components (organization or system imbalance).

A system is said to be *totally disorganized* if the system organization index B is at its maximum value of unity and *totally organized* if the index is at its minimum value of zero. NOTE that organization or disorganization do not carry positive or negative meaning, per se. Whether one desires a system to be organized or disorganized depends on the function the system is to perform. The change in the organization of a system over time may be taken as a measure of the change in performance of the system, with respect to its function, over time.

One further characteristic of B as it relates to the organization of a system is the following. It is easily proven that any function induces a natural partitioning of its domain into disjoint equivalence classes. Given the n components of a system S, one can define the state of the system S as the n-tuple of relative utilizations, $<u_1, \ldots, u_n>$. The relative utilization (power, work rate) state space is then

$$S = \{<u_1, \ldots, u_n> \mid \sum_{i=1}^{n} u_i = 1\}$$

which is the domain of the organization measure B. Two states will be in the same equivalence class if they both yield the same index value. Hence B discriminates states only with respect to the total amount of utilization (power, work rate) which must be redistributed for equal component loading. For example, in a three component system, <.4,.3,.3> and <.4,.32,.28> would both yield the same measure, B= .9.

3. APPLICATION AREAS

We indicate several potential application areas where the measure might be applied. *The examples that follow are not to be considered an exhaustion of potential application areas.*

3.1 System Performance and Tuning

The Measure B May Be Used As An Index of Subsystem Balance

Let the system S consist of three disjoint logical channels. Logical channel S_1 consisting of physical channels s_{11}, s_{12}, s_{13}; S_2 consisting of s_{21}, s_{22}; and S_3 consisting of s_{31}, s_{32}, s_{33}.

Suppose system monitoring (or prediction by other means) indicates the following observed physical channel utilizations.

$u'_{11} = .3$	$u'_{21} = .4$	$u'_{31} = .1$
$u'_{12} = .2$	$u'_{22} = .1$	$u'_{32} = .05$
$u'_{13} = .05$		$u'_{33} = .05$

and hence the following logical channel utilizations $u'_1 = .55$, $u'_2 = .5$, $u'_3 = .2$. The relative logical channel utilizations are calculated using Eq. (1).

$u_1 = .44 \qquad u_2 = .40 \qquad u_3 = .16$

The balance factor is calculated using Eq. (2),

$$b = (.44-1/3) + (.4-1/3) = .17$$

The organization index B, using Eq. (3), is then

$$\frac{3(1-.17)-1}{2}$$

or B= .74. Hence the system shows some bias or organization in its use of the logical channels. Seventeen percent of the total logical channel load could be redistributed.

If one were to measure the organization of the physical channels rather than the logical channels, (i.e., eight components), one would find

$$b = .345$$

$$B = .25$$

indicating a higher degree of organization with 34.5% of the total channel load candidate for potential redistribution.

If one were to measure the organization of the individual logical channels one would find

for s_1 b = .24
B = .64
for s_2 b = .3
B = .4
for s_3 b = .17
B = .74

indicating imbalance because of higher utilization of primary physical channels.

One valuable application of the measure is to show, quantitatively, improvements made in a system as a result of performance tuning efforts. Device banks could have been chosen in this example.

3.2 General System Investigation

The measure be used as an investigative tool for general system study. Suppose memory is partitioned into n regions. Calculating relative memory utilizations and then b and B may give insight into memory performance in a black box environment. Low B values may indicate some kind of priority handling or memory affinity is present.

Suppose a multiprocessor system is being studied and some attribute which can be associated with processors is chosen (e.g., CPU utilization in a given mode). Calculating relative processor utilizations in this mode and the corresponding values of b and B may again indicate some affinity for certain processors in that mode.

It is not our intent in this paper to compile a potential application list since the list depends to a great degree on the system studied. We do point out the importance of random sampling and its potential use in any application of the system organization index (see Ref. 2).

3.3 System Planning

Recent advances in computer system modeling have indicated that hardware/software utilization measurements may be predicted (derived) from functional relationships which depend on such parameters as:

1. the job attributes of jobs running on the system,

2. operating system module attributes,
3. hardware configuration parameters,
4. operating system philosophy as embodied in the operating system logic.

If we represent the first three sets of parameters indicated by n-tuples $\bar{X}_1, \bar{X}_2, \bar{X}_3$, a component relative utilization can then be represented by

$$u_i = F_i(\bar{X}_1, \bar{X}_2, \bar{X}_3)$$

The F_i's are algebraically formulated using the operating system logic and indicate how the operating system logic affects the component utilizations. Hence any system organization index representable as a function of component utilization, is now parameterized to factors controllable to some extent by the planning process; i.e., $B(u_1, \ldots, u_n) = B(F_1(\bar{X}_1, \bar{X}_2, \bar{X}_3), \ldots, F_n(\bar{X}_1, \bar{X}_2, \bar{X}_3))$. For example, the effect of changes in operating system parameters could be quantified more easily. The use of predictive modeling tools in conjunction with the organization index may be of value in the system planning process.

4. REMARKS ON FURTHER EXTENSION

The system organization measure has been defined using a particular criteria for the f_i in the definition of b, Eq. (2) in Section 2.

1. It would be of interest to study B from a more general perspective of organization using other criteria for total disorganization other than uniform loading of power (utilization) across components.
2. It was noted that the change in the organization of a system over time may be taken as a measure of the performance of the system with respect to its function over time. The definition of system performance measures based on the system organization measure approach appears to be a potentially fruitful path for further investigation.
3. The integration of the organization measure into current measurement system profiles should be investigated.
4. The role of the organization measure as one of the objective functions to be considered in capacity planning of systems should be considered.

REFERENCES

1. C. E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, Vol. VII, July 1948.
2. R. A. Orchard, "A New Methodology for Computer System Data Gathering", *Proceedings of Thirteenth Meeting of CPEUG*, 159-182, (October 11-14, 1977).

PRELIMINARY MEASURE OF C.mmp UNDER A SYNTHETIC LOAD*

Samuel H. Fuller**, Patrick F. McGehearty, and George Rolf

Carnegie-Mellon University
Pittsburgh, Pennsylvania 51213

This paper discusses the performance of C.mmp, a multiprocessor, under a synthetic load imposed by a remote terminal emulator (RTE). The primary measure of performance is average response time to an interactive request for service. An analytic model is used to predict response time. The model is modified as a function of load and the number of available processors. The actual measurements of C.mmp agreed, to within experimental error, with a simple central server model once three factors were incorporated in the model: (1) 210 milliseconds of operating system overhead for each request for service, (2) Bandwidth limitations in the link between C.mmp and the RTE, and (3) Precise details of the processor scheduling algorithm. Future work will consider using more complete synthetic jobs to better stress primary memory management, secondary storage scheduling, and operating system facilities.

1. Introduction

In this study, we explore possible performance bottlenecks and overheads in the Hydra Operating System on C.mmp. Since a multiprocessor such as C.mmp allows more opportunities for parallel execution than the traditional uniprocessor and thus has a potential for more complex interaction between processes, it is not unreasonable to assume that performance problems might be harder to detect and understand properly. To aid our understanding, we take an incremental approach to studying the system. We develop a simple, easy to understand model of C.mmp as an interactive system. An artificial load, tailored to match the model as closely as possible, is imposed on C.mmp. We compare the model predictions with actual performance and look for discrepancies. Simple configurations are studied first to ease the task of identifying the causes of the discrepancies. The model is modified to include these causes and other configurations are examined. As further discrepancies are found, their causes are also identified and included. In this preliminary study, we only examined the process scheduling behavior of the system. Program memory size was minimal as was use of operating system facilities.

2. SYSTEM DESCRIPTION

Figure 1 shows a PMS diagram of C.mmp at the time this study was done, April 1976. Five PDP 11/20's and three PDP 11/40's were in operation (as of March 1977, a full sixteen processor configuration was operational). Each I/O device was attached to an individual processor. For example, Pc [0], an 11/20, had a moving-head disk controller, a line clock and console teletype. The link to the front end processor (FE-link) on Pc[4] was time multiplexed among up to 20 terminal lines and has a maximum capacity of 60 characters per second. To impose a controlled, repeatable load, the normal front-end terminal handler was replaced by a Remote Terminal Emulator (RTE). RTEs have been used previously for similar studies on other systems. [Turner & Levy, Lassitere and Scherr]. The RTE allowed from 1 to 28 terminals to be simulated by means of a script. Although all terminals followed the same script, each terminal might be at a different place in the script. The RTE also had the capability for measuring time intervals between actions on a line.

Before measurements began, each virtual terminal on the RTE logged in and started a synthetic process which was told which set of processors to execute on. During the measurement phase, the following loop was executed on each terminal:

A. A think interval was drawn from the think time distribution. During this interval, the terminal was idle.

B. Next, a compute request was drawn from the compute time distribution. The request was sent to the synthetic process for that terminal. Timing began immediately after the sending of the carriage return was initiated.

C. When the RTE received the prompt indicating that the synthetic process had completed the requested computation, the delay was computed and recorded. This time interval is defined to be the response time.

The measurements continued until a preselected number of responses were recorded over the whole system. Internal timings of the RTE were also made to ensure that internal overheads did not distort results. After the measurement run completed, summary results were calculated and printed.

On C.mmp, a complex series of actions occurred for each compute request. To explain these actions, parts of the operating system will be described in detail. The Hydra kernel contains a round robin scheduler for allocating "feasible" user processes to the various processors. To be feasible, a process had to be allocated primary memory by a "Policy Module". The policy module used for these measurements merely allocated memory to requesting processes on a strict first come first served basis. It used two processes to carry out its task, WSTOP and PAGER. WSTOP received process-related event messages from the kernel in order to maintain its process-state tables, which it shared with PAGER. Whenever possible, PAGER scheduled processes to be swapped into primary memory. Another process, known as the terminal subsystem, provided communication between the processes and terminals. It passed messages between the individual processes and the FE-link.

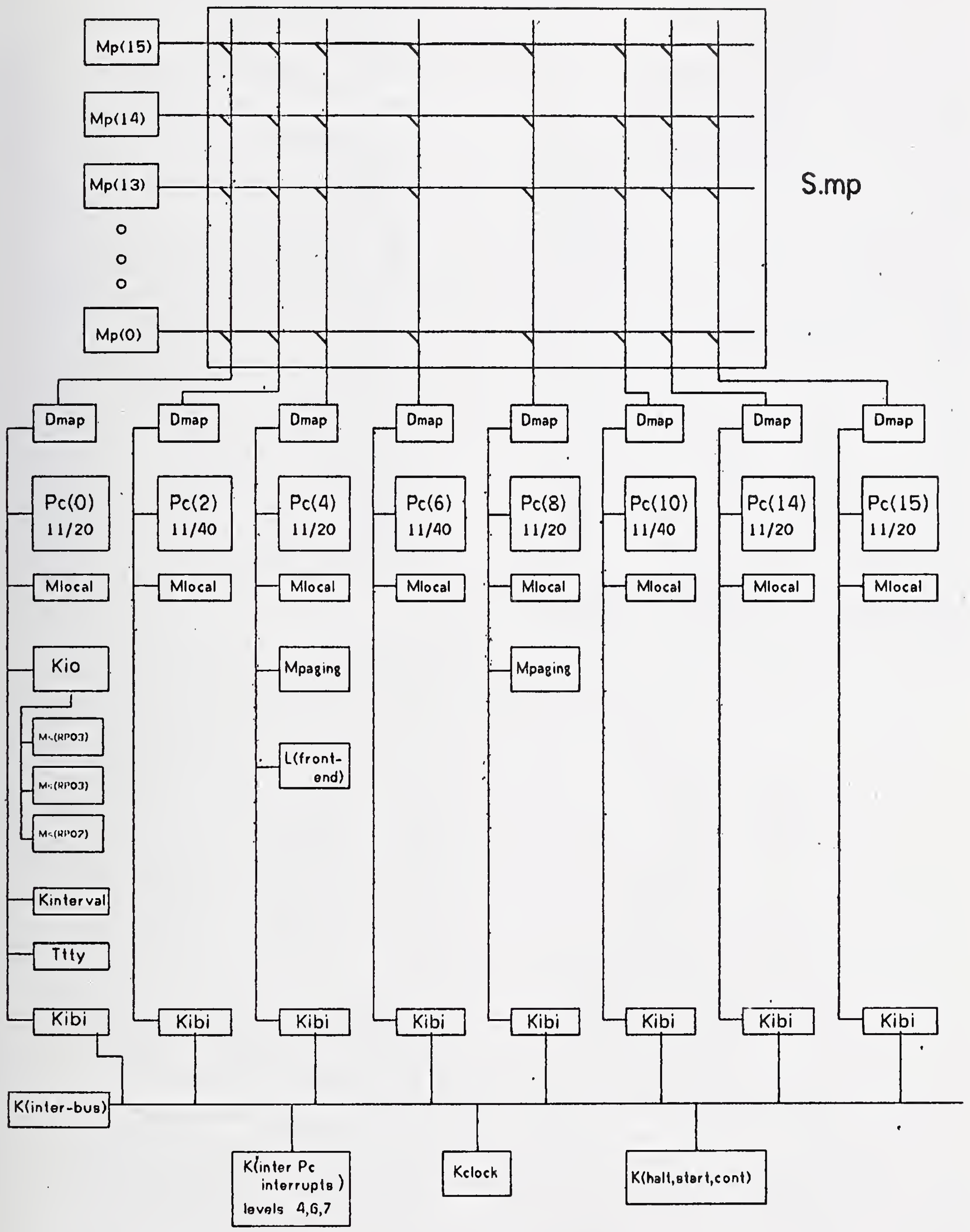


Figure 1. The eight processor configuration of C.mmp

Figure 2 gives a graphic representation of a single interaction. First, the kernel received the characters from the RTE over the FE-link (1). The characters are bufferea until a "break" character such as a carriage return is received. Then they are bundled in a message and passed to the port system in the kernel. WSTOP was notified by the kernel that a process requested service (2). WSTOP recorded the relevent information and notified PAGER that a change in the set of runnable processes had occurred (3). PAGER then placed the synthetic process in the feasible list (4). The synthetic process was swapped in core if necessary, and scheduled on a processor by the kernel (5). The synthetic process received the message from the port system, read it and acknowledged it (6,7). The requested computation was performed by executing a small compute loop a specified number of times (8,9). Table 1 shows the length of time a single computer loop would take on the various Pcs used in these measurements. After the synthetic process finished its compute request, it sent a prompt to the terminal system (10). The terminal subsystem passed the prompt to the kernel to send to the FE-link (11,12). Meanwhile, the synthetic process checked for more messages from the terminal. Since none were available, it blocked waiting for input (13). WSTOP is then notified of the new condition of the synthetic process by the kernel (14) and the interaction is complete. Obviously, a significant degree of overlap among the various processes occurs if several processors are available. However, if the system were heavily loaded, processors would not be available immediately upon request and the various processes would run sequentially.

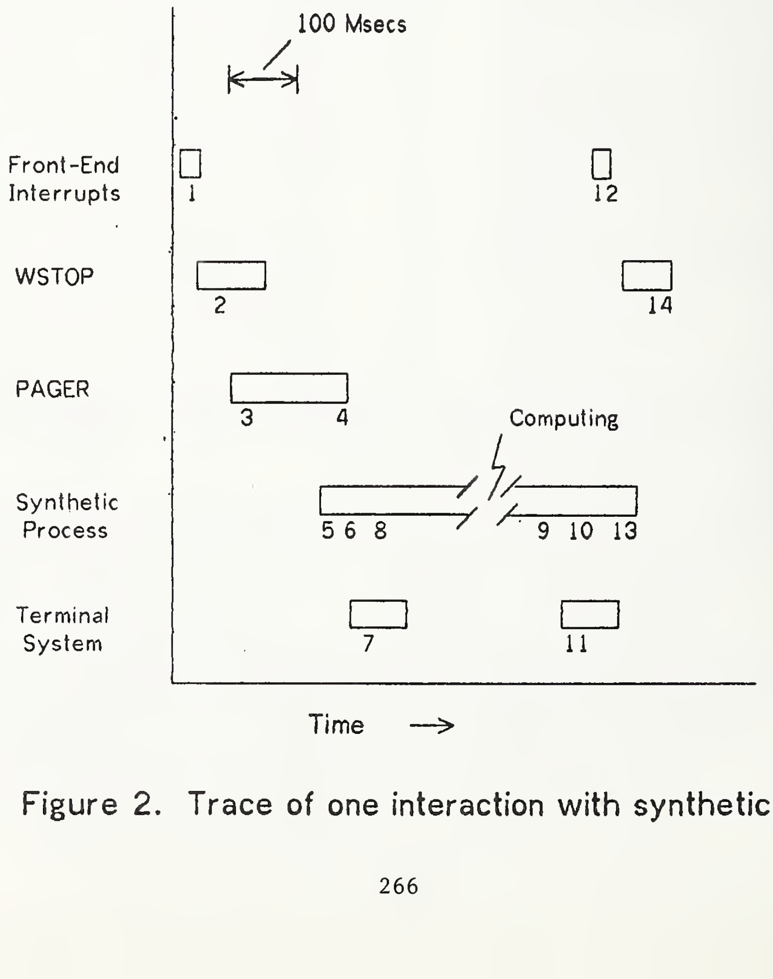


Figure 2. Trace of one interaction with synthetic job

3. PERFORMANCE CRITERIA

Since C.mmp/Hydra is intended to be used as a general purpose computer system, no single measure adequately captures the performance of the system. However, in this paper we concentrate on the interactive performance of the C.mmp/Hydra system and leave to other reports the consideration of other factors in the overall performance of C.mmp/Hydra [Fuller, 1976; Fuller and Oleinick 1976; McCracken 1977; Robertson and Newell, 1975; Fuller, Swan and Wulf, 1973].

Given our interest here in interactive performance, we will be using response time as our principle measure. In fact, in this preliminary study, we restrict ourselves to considering the average response time. We have a number of reasons for considering only average response time here:

A. In this study we are attempting to identify the primary performance characteristics of C.mmp/Hydra. As we consider variations in interactive load and number of processors, we will be using average response time as an indicator to identify when a performance problem occurs in the system.

B. We are using an analytic model as the standard against which to compare the actual performance and other measures of responsiveness (e.g., variance in response time, 90% point in response time, etc.) are more difficult to extract from an analytic model.

C. Average response time has often been used in the past and we would like to be able to compare our measures from C.mmp/Hydra with measurements from other time-sharing systems.

4. PRELIMINARY MEASUREMENTS

Before detailed response time measurements were made, some preliminary data was collected. Hydra has a trace facility for recording internal events such as process scheduling and kernel calls (similar to monitor calls in other systems). Using this facility, it was found that a "null" interaction consisting of receiving a request for zero computation and replying to it required approximately 210 milliseconds of 11/40 processor activity (See Figure 2). Since some of this activity could occur in parallel, and some occurred after a response is made, the minimum response time when three 11/40 processors were available was around 100 milliseconds. The measurement process imposes an overhead that is allowed for to some degree in the numbers mentioned above, but an error of +/- 10 milliseconds is still possible.

Several input parameters were fixed for all runs to reduce sources of noise. All runs had a least 625 interactions per run to provide acceptable confidence intervals, based on our pilot runs. The 90% confidence intervals are marked on the accompanying figures. All runs used the same samples from distributions chosen so that the mean think time was 5 seconds and (for all except the minimum load runs) the mean compute time was equivalent to 2.1 seconds on a PDP 11/40 or 3.2 seconds on a PDP 11/20. The preceding numbers were chosen to impose a moderate load on the tested configuration. Since we were limited by Hydra internal table sizes to 20 terminals, we used the relatively short think time of 5 seconds instead of the more usual 30 to 35 seconds [Lassitere and Sherr] in order to provide a higher interaction rate. To minimize effects of the terminal properties, the characters for the compute request were started 0.3 seconds before the end of the think time. In this way, processing could begin immediately after the carriage return was sent.

5. MODELS and MEASUREMENTS

Figure 3 shows our initial model of the C.mmp system. Since the number of terminals equals the number of processes in the system, there is no queueing at the RTE. The model includes the fact that the different processors run at different rates. It is assumed that processes are scheduled on the faster processors (i.e., the PDP-11/40) if they are available. If all service times are exponential, Jackson's Theorem [Kleinrock Vol. 1; Buzen 1973] yields convenient solutions of the analytic model. Note that the model does not explicitly represent the various stages of computation on C.mmp. In particular, the model does not allow for parallel processing. When the system is lightly loaded, most system processing occurs in parallel to user processing,. Therefore, the model should predict slightly longer response times than those measured in the system under light loads. We did not conisder this discrepancy to be serious for two reasons. First, the maximum overlap is on the order of 10% for the loads which we examined. Second, we are primarily interested in system overheads and bottlenecks when the system is near saturation.

Next, we want to impose a load on the system that matches the model as closely as possible. Note that the model assumes exponential service times for the terminals and C.mmp. As was pointed out previously, the minimum compute time is significantly greater than zero. This characteristic prevents any sequence of compute requests made by the RTE from having an exponential distribution. However, we approximated an exponential distribution by choosing either a compute request of zero with a probability of x or a compute request from an exponential distribution that has a mean of y with a probability of 1-x. These compute requests are in addition to the minimum compute time, t. To achieve a distribution with a mean and standard deviation of z from the combined distribution, x and y must be as determined by eqns (1), (2) and (3).

$$x = p(2-p) \qquad (1)$$

$$y = z(1-p) / p(2-p) \qquad (2)$$

where,

$$p = t/z \qquad (3)$$

Note that as p goes to 0, x goes to p and y goes to z. When p is 0.1 as is true in our case, x and y are very close to p and z.

5.1 Case 1: Negligible Compute Load

To find out what delays would occur in a load with light compute requests, runs were made with zero compute requests (The results are shown in Figure 4). The non-zero response time for one terminal can be attributed to the overhead for handling a compute request. Note that the initial model predicted no significant increase in response time as the number of terminals increases while the system showed a noticeable increase.

The dominant cause of this discrepancy was the Front-End (FE) link bottleneck. This link was at 65% of capacity

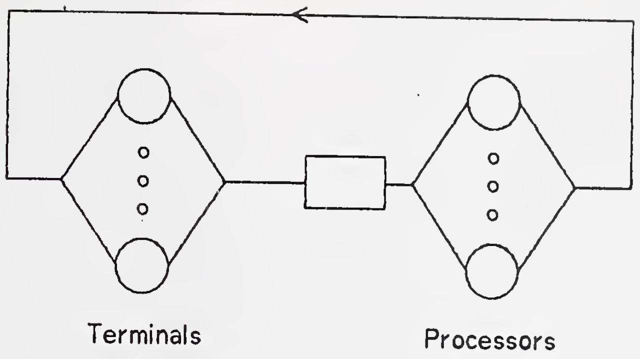


Figure 3. Initial Model of C.mmp/ Hydra

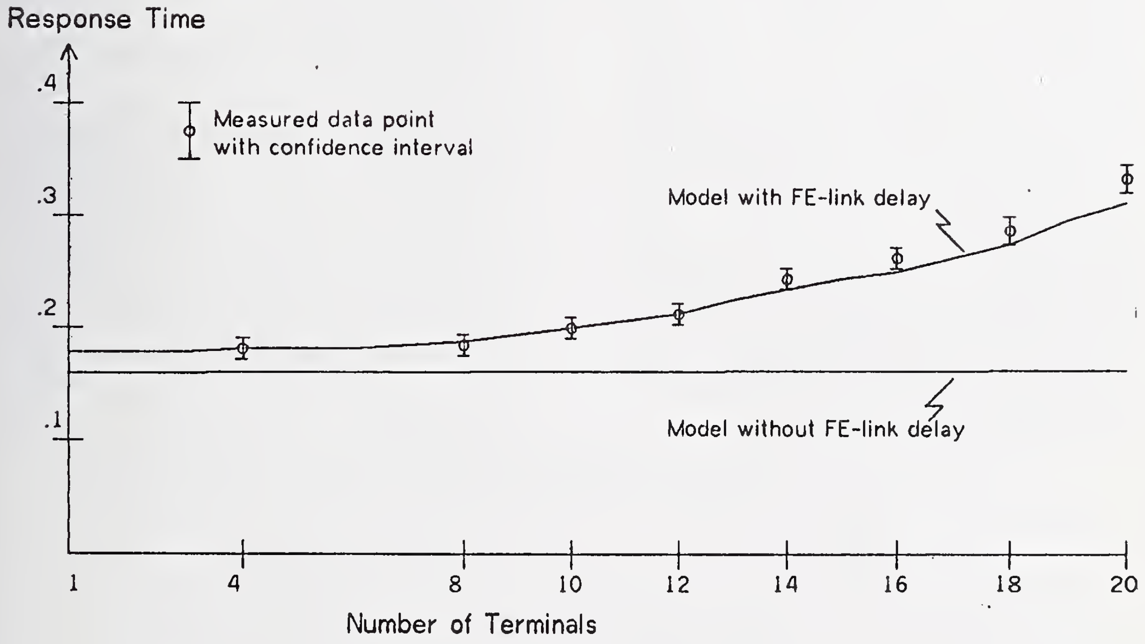


Figure 4. Five PDP-11/20s with no compute load

when each of 20 terminals was interacting once every five seconds. (I.e., a combined rate of 4 interactions/second). To incorporate this factor, we developed a second, more detailed model which included the FE link as a server (see Figure 5). The results of this second model are also shown in Figure 4. This model was a much better predictor of response time. The cause of the remaining slight discrepancies cannot be determined because the potential error in the input to the model is large relative to the discrepancies observed.

5.2 Case II: Uniprocessor configuration of C.mmp/Hydra

Next, we attempted to measure C.mmp as a uniprocessor. The results are shown in Figure 6. The synthetic processes were restricted to one 11/40. Unfortunately, the 11/20s could not be configured out of the system and the operating system processes continued to use them for their computation. Although the model does not include this inaccuracy, the predictions are within the 90% confidence intervals of those data points measured. Since the 11/40 was completely saturated with 4 terminals, no further measurements were made for this configuration.

5.3 Case III: The five PDP-11/20 configuration

Since no modifications to the model were necessary from the uniprocessor case, we moved on to a multiprocessor system. The configuration of five 11/20s and no 11/40s was chosen to study Hydra when processor speeds were relatively uniform. Figure 7 shows the results of the measurements. All model predictions are within the 90% confidence intervals of the measured data. There is a slight trend for the model to underestimate as the number of terminals increase, but the error factor in the measurement process makes such trends difficult to interpret.

5.4 Case IV: The eight processor configuration

The final configuration of this study was the maximum system at the time of measurements, five PDP-11/20s and three PDP-11/40s. When 11/40s and 11/20s were both available, a process was scheduled on an 11/40 at dispatch time. The analytic model significantly underestimates the measured response time (see Figure 8). Further investigation of the scheduling policy revealed that if a process is running on an 11/20 and an 11/40 becomes available, the process is not

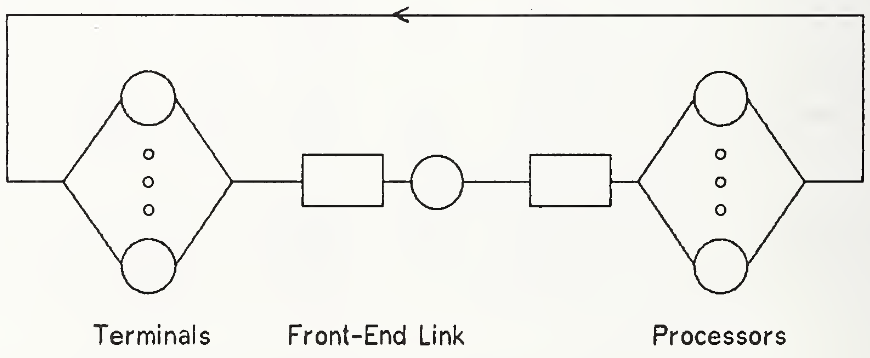


Figure 5. Detailed Model of C.mmp/ Hydra

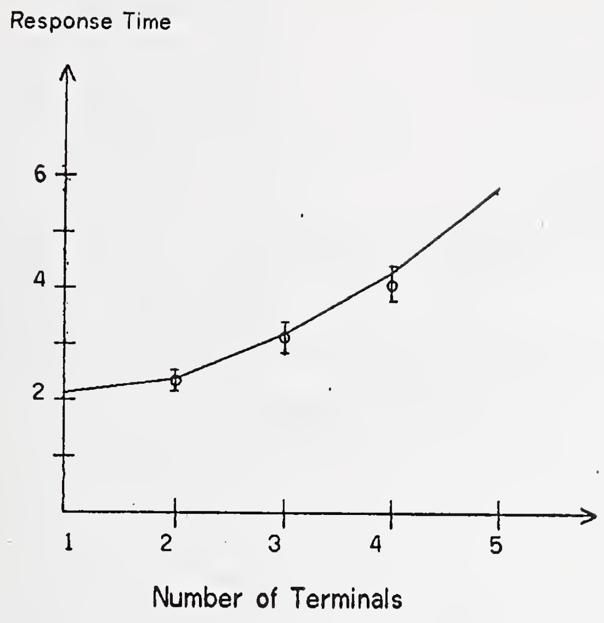


Figure 6. One PDP 11/40 with normal compute load

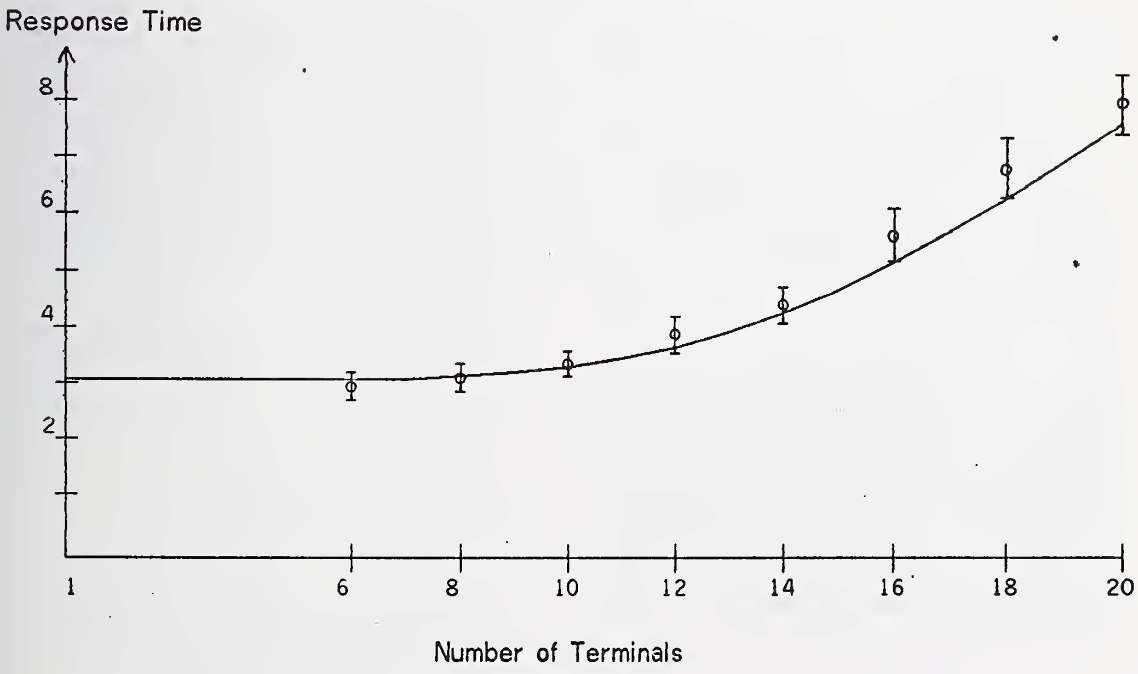


Figure 7. Five PDP 11/20s with normal compute load

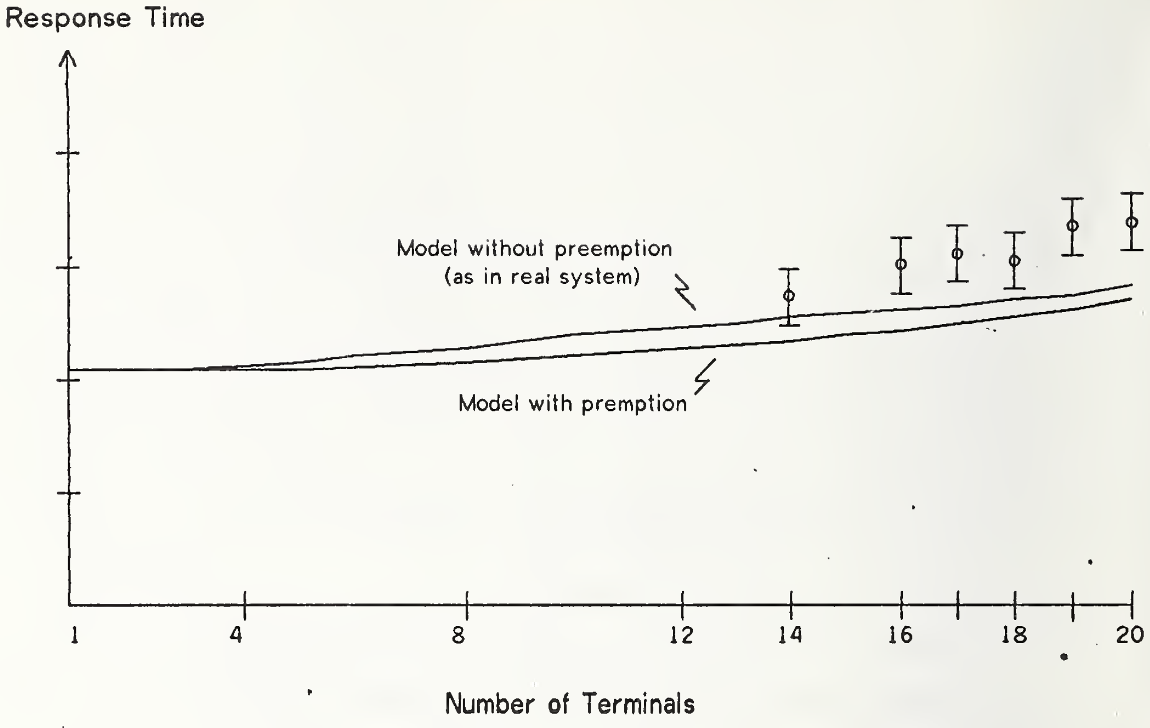


Figure 8. Eight processors with normal compute load

transferred to the 11/40. The analytic model assumes that it is. Another Markovian model [W. Corwin] that reflects the actual scheduling policy was developed and plotted on Figure 9 (labeled without preemption Model). Since the time of these measurements, the scheduling policy has been revised so that an idle 11/40 will preempt an 11/20. Even the nopreemption model underestimates the response time. No single cause of this remaining discrpancy has been identified at this time. We intend to instrument C.mmp/Hydra in more detail in order to make intelligent hypotheses as to its cause(s). Then we will devise experiments to test these hypotheses.

6. CONCLUSIONS AND COMMENTS

The main value of the RTE measurements reported here were to confirm that C.mmp/Hydra, in its early stages of operation, does in fact handle a simple interactive work load without experiencing any major performance anomolies. In fact, while it is difficult to quantify, one of the major values of the RTE emulator has been to repeatedly put heavier interactive loads on C.mmp/Hydra than are experienced under operational situations and hence uncover bugs and performance problems before they become a serious problem to the user community. Our future goals for the RTE reported on here is to extend its capabilities so that its synthetic load makes much more extensive use of Hydra's facilities, primary memory, and secondary storage so that it can better detect remaining performance problems in the C.mmp/Hydra system.

With relatively simple models and simple synthetic jobs, we identified several performance discrepancies. These discrepancies in turn lead to a better understanding of Hydra. Indeed, the scheduler has been improved since the time of these measurements to recognize idle 11/40s and move processes onto them from 11/20s. It should be noted that the synthetic load was much heavier than normal user loads and thus was more prone to cause synchronization bugs to surface. Several elusive bugs were tracked down because the RTE was able to replicate the conditions causing them.

One must not accept the results of any such system without critically studying all assumptions underlying the model. In this study, memory contention and memory scheduling were explicitly avoided. The synthetic jobs did not use the I/O systen, the File system or the operating system tables as much as real user programs are expected to use these facilties. Measurements of users of C.mmp/Hydra are needed to find realistic distributions of think and compute time. The system might also be studied under the stress of many more processes. A simulation model is being developed to more accurately reflect the internal workings of Hydra to see what other factors affect system performance. The possible causes of the remaining discrepancies between actual and predicted response times will be investigated and studied.

References

1. Fuller, S.H.,and P.N. Oleinik, "Initial measurements of parallel programs of a multiprocessor," 13th IEEE Computer Soc. Int. Conf., Washington, D.C., Sept. 1976.

2 Fuller, S.H., "Price/performance comparison of C.mmp and the PDP-10", IEEE/ACM Symp. on Computer Architecture, pp. 195-202, Jan. 1976.

3. Klemrock, L. Queueing Systems, Volume 1: Theory, Wiley & Sons, New York, 1975.

4. McCraken, D., "A production system version of the Hearsay 11 speech understanding system", PhD Dissertation, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213, 1977.

* This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (contract F44620-73-C-0074) and is monitored by the Air Force Office of Scientific Research.

** S.H. Fuller is now at Digital Equipment Corp., 1925 Andover Street, Tewksbury, Massachusetts 01876

PERFORMANCE STUDY OF A MINICOMPUTER SYSTEM

S. K. Lee
R. B. Maguire
L. R. Symes

Department of Computer Science
University of Regina
Regina, Saskatchewan S4S 0A2

The paper describes a study carried out to measure the performance of a time-shared minicomputer system. The system is a PDP-11/34 based RSTS/E system used in an undergraduate Computer Science laboratory. Results of the study have shown that in such an environment a relatively inexpensive minicomputer facility can support a large number of on-line users. Specifically, the study indicated how to expand the system from its current thirty to forty or more terminals through software modifications and minor hardware expansion.

The measurement package is internally driven, thus eliminating the need for human intervention or a front-end computer. The measurement system creates a number of pseudo terminal users. Activities of these pseudo users are controlled on an individual basis and an event trace of all activity is recorded. The technique used requires no external resources and permits the type of load to be varied conveniently. The measurement system incurs little system overhead, thereby giving an accurate characterization of system performance.

Key words: Minicomputer system; performance measurement; response time; time-sharing; workload.

1. Motivation

Our usage of a time shared general purpose system in an undergraduate Computer Science laboratory has been steadily increasing during the past several years. The PDP11/34 minicomputer system is used during the school year by approximately four hundred students, most of them in their first and second year of university with little or no programming experience. The reason for assigning these students to such a system instead of the large central computer at the University is to provide direct exposure to hardware. Although the system is satisfactorily handling the current load, it was uncertain how much the load could be increased before response time became intolerable. Consequently a study was initiated to determine the performance characteristics of the system and to provide information on how to bring about improvement in the most cost effective way.

2. System Configuration

The system is a PDP11/34 running the RSTS/E version 6B time sharing system with 124K 16-bit words of semiconductor memory, three RK05 moving head disks (2.5 MB each), a dual drive floppy disk and a mixture of thirty terminals.

3. Measurment

A software measurement package was developed to perform the functions of system stimulation, measure response time and collect system statistics. The package was driven internally by a number of pseudo terminal users, each from a different pseudo keyboard (a special and often very useful feature available on RSTS/E V06B). A pseudo keyboard is a non-physical device that has the characteristics of a terminal and is treated by the rest of the system in the same way as a real terminal. The workload imposed by each user and the characteristics of each user (e.g. think time between commands and typing speed) are built into the prepared scripts. A set of such scripts was constructed following an observation period in which typical users of the system were timed and their output collected. Programs contained in the scripts were selected from those obtained during the observation period. Combinations of these scripts were used to represent various types of workloads. A sample script is given in Figure 1.

The number of pseudo users (i.e. scripts) is specified at the beginning of each measurement session. This allows the load level and user mix to be varied conveniently. Starting time, as well as the duration of each measurement run, can be preset to a particular hour of the day. Since no more operator intervention is required after the initial dialogue, automatic initiation can be preset to occur during unmanned hours. This avoids interference with, and from, normal operation and facilitates sleep.

To ensure that the scripts matched the real workload, the package was used to collect statistics during normal operation. These results were compared to the results obtained running the scripts. The differences were minimal.

During a measurement run, activities of each pseudo user are controlled on an individual basis. A time-event trace of all or selective terminal users is kept in a log file and response time is recorded for each transaction. While this is in progress, a set of statistics collection routines is invoked periodically to monitor system activities such as CPU utilization and disk status.

The measurement package written in BASIC-PLUS consists of two parts. The first part is a multi-pseudo terminal driver. In order to minimize the amount of code in memory at any one time, this part is subdivided into two routines (R-1, R-2). The first routine (R-1) is responsible for initiating dialogue during start-up, initializing run parameters, processing script files and creating a virtual array for storage of the script lines. When the execution of this routine is completed, it "chains" to the second routine (R-2) to start the measurement run. By "chaining", the code for the first routine is swapped out of main memory and the code for the second routine is brought in. Thus, only the code needed for the current phase is in memory. R-2 controls the activities at each of the pseudo terminals. It determines when and what to input to the system. It monitors the output from the system, determines if it is a response or just an echo, maintains a time-event trace and records the response time for each input line. To keep the response time measurement accurate to within 1 second, each terminal in the wait-for-response state is polled at the beginning of every second for system output. During each polling cycle, R-2 is locked in memory, thus preventing it from being swapped out. At the end of the run period all pseudo users are logged off the system, files are closed and R-2 removes itself from the system job list.

The other part of the package is made up of a set of programs to collect, compile and summarize data on system component activities. Once activated, routines are invoked periodically to take snap shots of the system. A report summarizing the statistics on CPU usage, disk usage, run time statistics, file processor status, etc. is produced at the end of the measurement run.

4. Measurement Results

As mentioned above, a set of scripts each representing typical workloads from students with varying degrees of programming experience was prepared. Some scripts contain frequently used commands for login, logoff, file building, simple editing and running small BASIC programs, tutorials and games. Others make greater use of the Fortran compiler, linker, editor, file manipulator, macro assembler and larger programs. With these scripts, the workload can be made to match different running environments. Four distinct types of workloads were created to simulate observed situations:

(1) mixed-user type - This represents an open laboratory environment, that is, lab usage during unscheduled hours for course assignments. In those hours, the user group is a mix of the students described in 2 below.

(2) single-user type - This represents scheduled laboratory usage by students in a particular class. This type can be further divided into three subclasses: introductory and advanced programming laboratory periods and laboratory periods running only tutorials. In this paper, only the results for the advanced programming laboratories are presented because they

```
!C               ADVANCED USER SCRIPT
!C       This script simulates the workload of a 300 student.
!C       F300A.FOR,F300B.FOR and F300C.FOR are routines to be
!C       linked together to form an electronic calculator program.
!C
!C       The following explains the use of script directives :
!C         !C              comment.
!C         !P              enter system output into log file.
!C         !NP             discard system output.
!C         !LOOP n         repeat the script lines up to next !ENDLOOP
!C                         n times.
!C         !ENDLOOP        end of loop.
!C         !THINK m,n      user think time for the following commands
!C                         lies between m sec and n sec.
!C         !PASSWD pj,pg substitute this line with the password
!C                         for account [pj,pg].
!C
!THINK 2,5
I50/50
!PASSWD 50,50
CC
!LOOP 2
!P
!THINK 5,30
EDIT F300A.FOR
10AL
!NP
EX
CC
PIP F300A.BAK/DE
!THINK 5,10
FORT F300A,KB:=F300A
FORT F300B=F300B
FORT F300C=F300C
!P
LINK CAL=F300A,F300B,F300C/F
RUN CAL
1+(2.2*(3.3-4.4)/5.5)=
5.6*102.3/22=
!NP
C
DIR
!P
RUN$PIP
!NP
KB:=F300C.FOR
D.FOR=F300A.FOR
D.FOR/DE
OC
!THINK 5,20
EDIT F300B.FOR
I
C      COMMENT
OJ
L-AK
EX
OC
!ENDLOOP
!THINK 2,5
PIP F300B.BAK/DE
OCCC
BYEF
!END
```

Figure 1 A Sample Script

represent the worst case among all the user groups.

(3) A group of users all running highly I/O bound jobs.

(4) A group of users all running highly compute bound jobs.

The last two are extreme cases, very unlikely to happen during normal operation. They are included only for comparison purposes.

Results obtained for the mixed-user type situation are summarized in Figures 2 to 6 inclusive. Figure 2 shows the average response time for 14 listed commands with the number of simulated on-line users ranging from 5 to 28. While response to most commands (including others often used but not shown in Figure 2) remained reasonably small over the entire range, a few increased drastically beyond the 15-user level. For example, Fortran users (see FORT, LINK, PIP and EDIT commands) find the response time unreasonable above the 15 user mark.

Figure 3 displays the average response time for all commands in the mixed-user environment, broken down by user type. It shows that even with the presence of some advanced users, introductory Basic and tutorial users are relatively unaffected by the load increase. But degradation was felt much sooner by advanced users. Owing to this we must restrict laboratory section sizes for classes using Fortran and limit the number of students using the laboratory for Fortran assignments. Consequently we have kept one of our multi-section Fortran classes on the University's computing facility.

Figures 4, 5 and 6 summarize the statistics collected. Figure 4 reveals that the degradation was caused mainly by the rapid increase in time lost waiting for I/O. Time lost means time when the CPU is available but must wait for the completion of disk I/O. As a result, the net amount of useful work done remained unchanged beyond the 15 user point. Beyond this point the system is I/O limited. Figures 5 and 6 further identify swapping as the major factor causing the increased I/O wait time. This implies that unless the amount of swapping can be decreased and/or the swapping rate can be increased, response cannot be improved significantly.

Results from the other three workload types confirm the above finding. A test simulating a group of advanced students (Figure 7) showed that the system was saturated by only a small number of these users. It was again limited by time lost in I/O wait.

In a heavily I/O bound environment (scripts continuously requesting disk file transfers) system saturation occurs even earlier (Figure 8). The compute bound environment tested the maximum capacity of the CPU when not limited by I/O wait. The scripts continuously computed complicated arithmetic expressions without intervening I/O. In this case the time lost due to I/O wait is insignificant. Compare figures 6, 7, 9 and 10. As a result, almost 95% of the CPU was allocated to the execution of user code.

5. Analysis

Having identified the major bottle necks in the system, we then considered a number of possible alternatives for upgrading the system to meet an increased demand. One way was to invest in a completely new, faster and more powerful machine like the PDP11/70. The disadvantage of this course was the high cost. Another way was to acquire another PDP11/34. The two machines could share most of the slower I/O devices as well as the workload. The third and probably the least expensive alternative was to optimize/upgrade the present system so that it could perform at its maximum capacity for the intended application. With this alternative, there are again a number of hardware and software options. Some of the hardware options are as follows:

(1) addition of a fixed head disk for swapping.

(2) an even faster swapping device such as a semiconductor fixed head disk replacement,

(3) a hardware memory cache, or

(4) a hardware floating point unit to improve computation time.

The system performance can further be enhanced by software tuning. An example of such 'tuning' is setting an optimum value for system parameters such as the number of small buffers (used for terminal I/O), the amount of software cache, (temporary storage for disk directories recently used - therefore most likely to be used in the near future), maximum job size etc.

The options that were readily available to us were to (1) install a fixed head disk and (2) allocate more buffer (cache) space to disk directories. With the fixed head disk installed, the system was measured again with the mixed-user environment. The results are summarized in Figures 11 to 15. These can be compared with the results shown in Figures 2-6 respectively. As expected, the system response greatly improved. At the maximum load the average response decreased by almost

50% (Figure 12 vs Figure 3). Response time for most of the worst case commands shown in Figure 2 dropped by as much as 75% (Figure 11). The improvement was the result of a significant increase in the block transfer rate during swapping, thus reducing the amount of disk wait (Figure 13-15). Useful work done by the CPU was increased by more than 60% (Figures 4 and 13).

A test was performed to study the effect of software cache on the system by disabling the cache (normally enabled) during an open laboratory period with about 15 real users on the system. Soon after the cache was disabled, system response became significantly slower than before (see Table 16 for the comparison). Activities at the terminals suddenly dropped owing to more wait time. Users, none of them having knowledge of the test, became very impatient. The test showed that the presence of the cache indeed has a significant effect on system response. (The size of the cache can be varied only by system regeneration. The effect of this is currently being investigated.)

6. Conclusion

The study demonstrates the effectiveness of internally driven measuring techniques. They have proven to be very flexible and convenient to use and require no external resources such as human involvement or a front-end computer. The overhead incurred was reasonably small, making it a very useful tool in system characterization. The study also showed that a relatively inexpensive time sharing facility can adequately accommodate large numbers of introductory and tutorial users, but only limited numbers of advanced users.

The work was supported in part by National Research Council Grant A-7912.

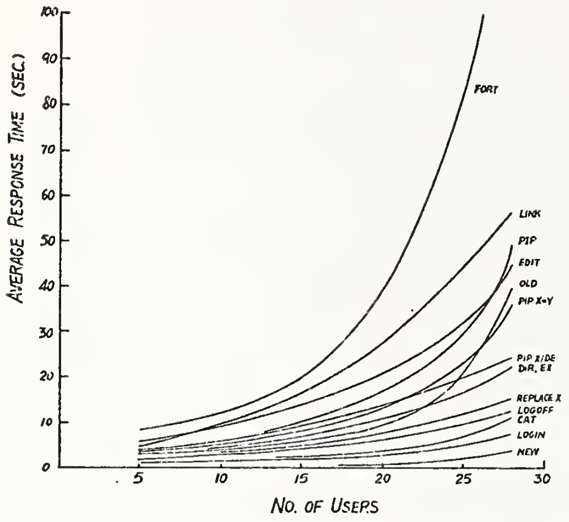


Figure 2. Average Response Time to Individual Commands in Mixed-User Environment

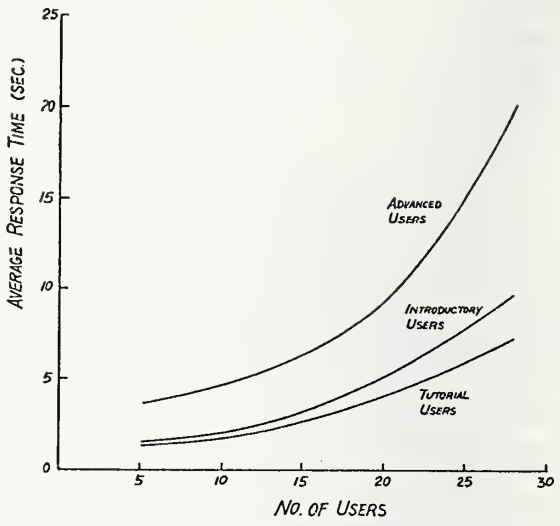


Figure 3. Average Response Time by User Group in Mixed-User Environment

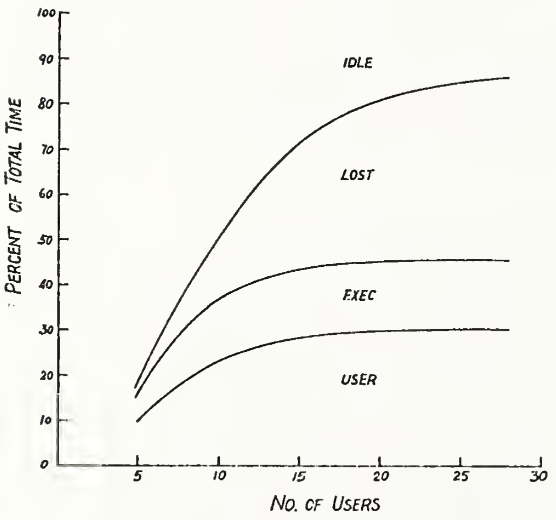


Figure 4. Run Time Statistics in Mixed-User Environment

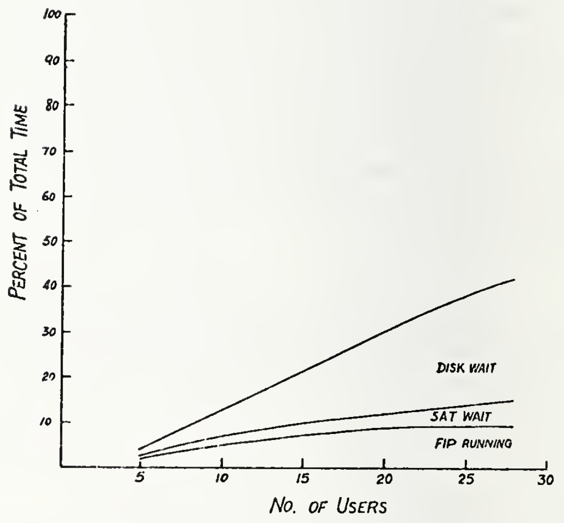


Figure 5. File Processor (FIP) Statistics in Mixed-User Environment

Note: Disk Wait - FIP waiting for disk file data
SAT Wait - FIP waiting for disk Storage Allocation Table.

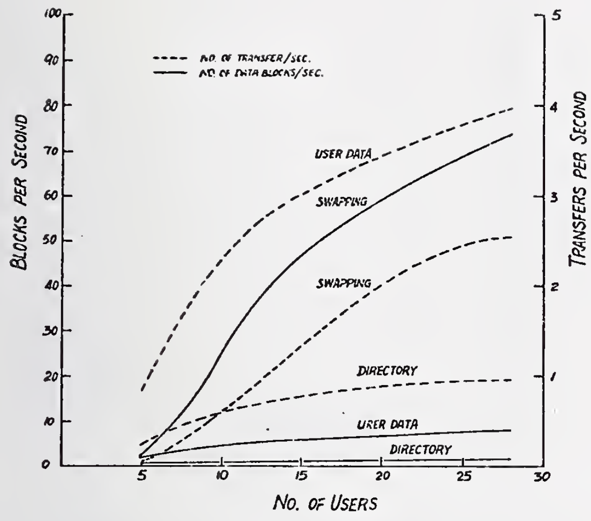


Figure 6. Disk Status in Mixed-User Environment

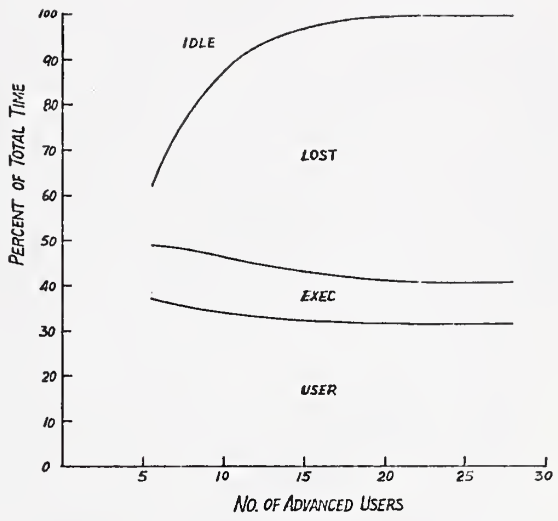


Figure 7. Advanced-User Type Run Time Statistics

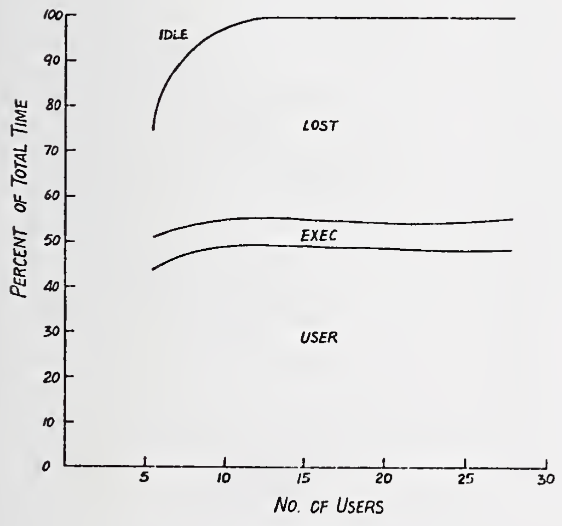


Figure 8. Run Time Statistics in Heavily I/O Bound Environment

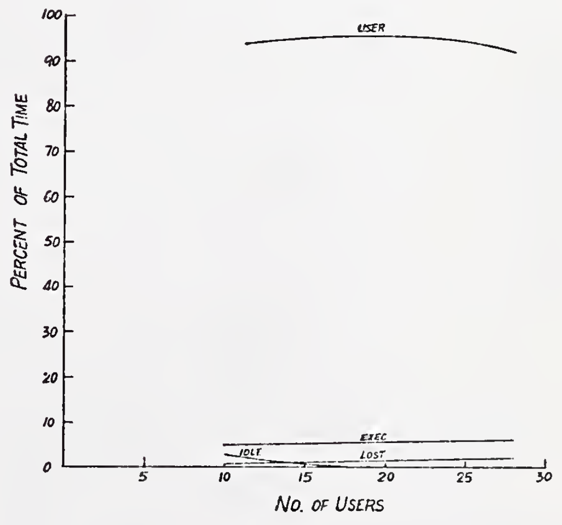


Figure 9. Run Time Statistics in Heavily Compute Bound Environment

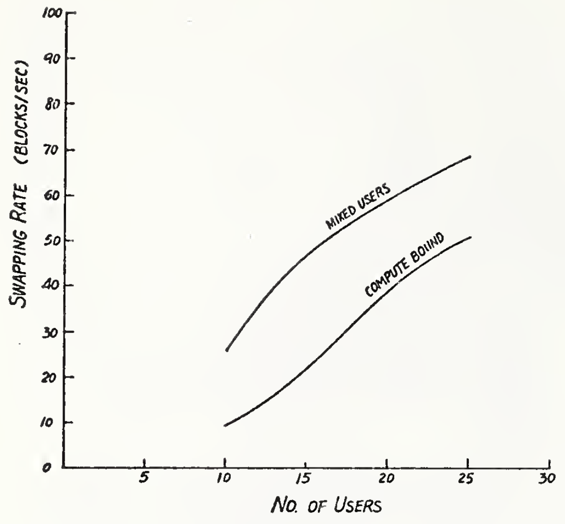


Figure 10. Comparison of Swapping Rate between Mixed-User Load and Compute Bound Load

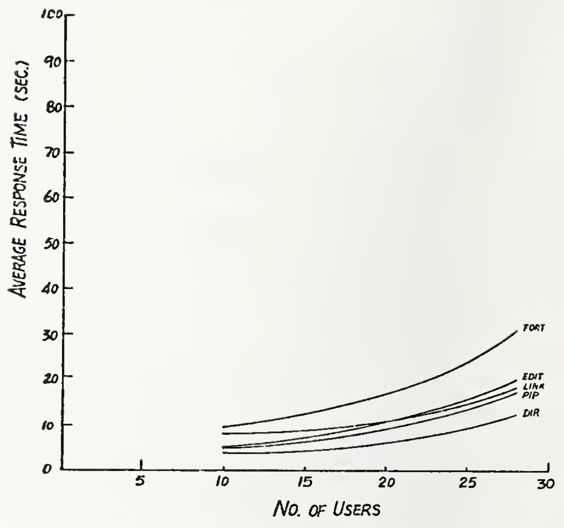


Figure 11. Average Response to Individual Commands in Mixed-User Environment with Fixed Head Disk

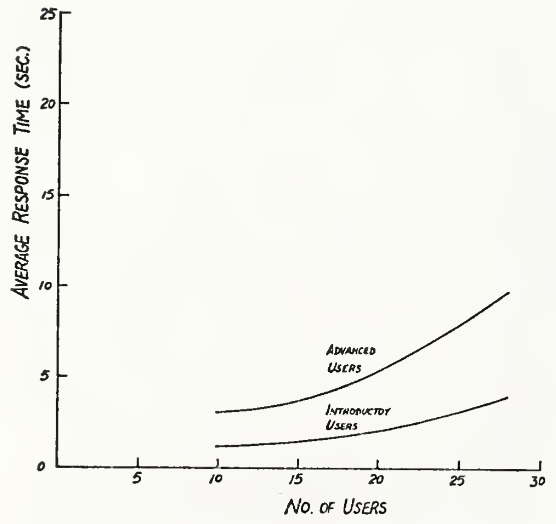


Figure 12. Average Response Time by User Group in Mixed-User Environment with Fixed Head Disk

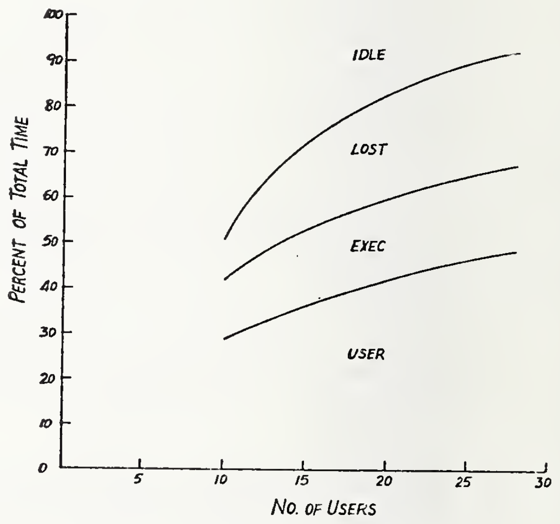


Figure 13. Run Time Statistics in Mixed-User Environment with Fixed Head Disk

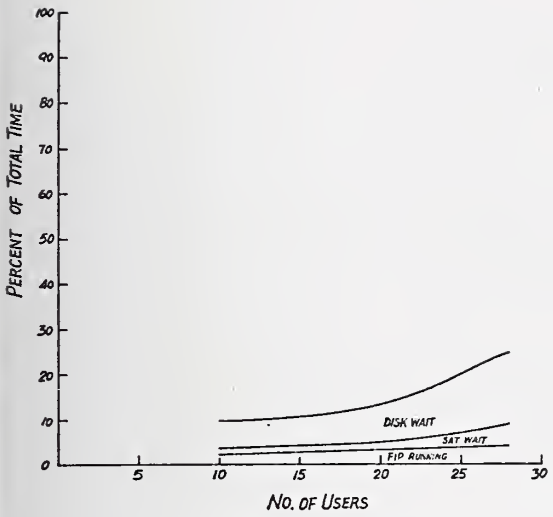


Figure 14. File Processor Statistics in Mixed-User Environment with Fixed Head Disk

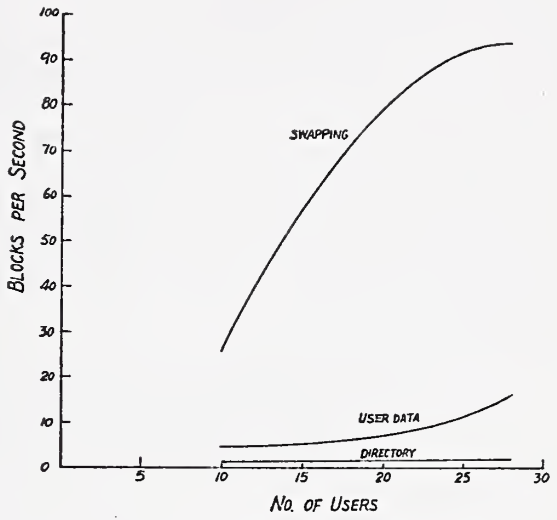


Figure 15. Disk Status in Mixed-User Environment with Fixed Mead Disk

TABLE 1
Comparison between no cache and with cache

	With Cache	Without Cache
Cache/FIP statistics =		
% hits	75.82%	
Cache CPU usage	1.29%	0.00%
FIP desired	21.35%	63.13%
FIP waiting	18.37%	60.03%
CPU status =		
user job	15.58%	25.95%
CPU idle	67.22%	56.50%
Monitor	11.74%	12.60%
Fast I/O	4.93%	4.87%
Disk Status =	BLK/sec xfer/sec	
user data	2.44, 0.83	2.62, 1.16
directory	2.36, 2.36	8.45, 8.45
swapping	21.70, 0.90	19.45, .91

TO MULTIPROCESS OR NOT TO MULTIPROCESS

Melvin Lieberman

Manager of Computer System Measurement
Chase Manhattan Bank
1 New York Plaza
New York, N.Y. 10015

A methodology is provided for deciding whether or not to combine two individual central processors as a multiprocessor. This methodology is sensitive to the following:

1. Number of terminals in network
2. Hardware availability
3. Software availability
4. Thruput differences
5. Staffing impact
6. Cost differences

Analysis of a typical configuration reveals that from a hardware point of view a multiprocessor can be expected to offer improved availability; but software availability is lower in the multiprocessor. The expectation of improved thruput that could not have been obtained from two individual processors does not appear justified. This occurs because the primary thruput benefit of a multiprocessor, reduction in segmentation of CPU and I/O capacity, is reduced at moderate utilization levels and the coupling loss resulting from CPU/CPU interferance for shared resources increases. Staffing patterns are not particularly sensitive to CPU population, therefore, changes resulting from implementation of a multiprocessor are not large.

In summary, the pre-tax, current value, cost differences did not, in this typical case, justify investing in the equipment to convert two individual processors to a multiprocessor.

Key words: Hardware availability; multiprocessing; present discounted value; software availability; technological matrix; thruput.

Presented here is an analysis employing classical industrial engineering techniques such as availability modeling and the machine interference model to assist in deciding whether or not to build a multiprocessor from two individual CPU's. Three factors are considered in the decision frame-work:

1. Impact on availability

2. Impact on site thruput

3. Impact on cost

Availability Overview

Multiprocessors (MP's), it is claimed, offer improved availability of the computing facility because one processor can take over for the other if it fails. This is a difficult benefit to establish. For the most part, commercial users are more concerned with delivering output at specified deadlines. A good measurement attribute for the ability to function well in such an environment would be the confidence in timely delivery. MP's don't give deliverability much of a boost. They function so as to give you part of your output by the deadline and the rest later. The remaining uniprocessor (UP) of a two UP installation would have done the same with a slight time-lag. Our methodology enables commercial users to assign a financial return to the availability improvement aspects of an MP.

Site Thruput Overview

MP's, it is felt, enchance site thruput. They may, but in very limited circumstances. Our analysis indicates that the value of an MP, in terms of increased thruput, is highest when it is least likely to be bene-ficial, i.e., when system utilization is low. Systems that have reduced thruput as a result of CPU vs. I/O segmentation should be tuned prior to examining the MP/UP decision. In either case, as CPU utilization rises, during peaks or across time, the probability of increased thruput declines, and eventually at 60% to 70% CPU utilization depending on coupling losses, the MP becomes less productive, in a thruput sense, than two UP's.

Cost Overview

It costs more money for a multiprocessor, generally speaking, than for the two individual processors which comprise it. This is a result of the requirement for an interconnect box which some vendors explicitly market or the interconnection circuitry distributed throughout the architecture of other vendor offerings. An explicit marketing approach is quite equitable, only those who want an MP need pay for it. Other vendors charge for it, whether or not it is used.

With regard to financial benefits, an MP may cost somewhat less money to operate than two UP's because fewer operators are required. It will reduce losses associated with terminal operator salaries which continue to accumulate when the host CPU programs are out of service because of hardware failure; however, the effects of software failure have a tendency to negate this cost reduction. Also, some core and disk savings are possible.

The techniques described in this paper enable users to prepare models that, in effect, allow them to financially test the usefullness of equipment without having to assume the risks of implementation.

Availability Details

Five system elements should be analyzed to estimate the impact of availability on the financial results of building an MP (component availability interrelationships are shown in Chart I).

1. Terminal Environment

2. CPU's

3. Operating System

4. MP Interconnecting Box

5. System Support Disks

Terminal Environment

Our model is based on an average of 200 operational terminals during a 4000 hour work year consisting of 250, sixteen hour days. Each terminal operator is valued at $10.00 per hour.

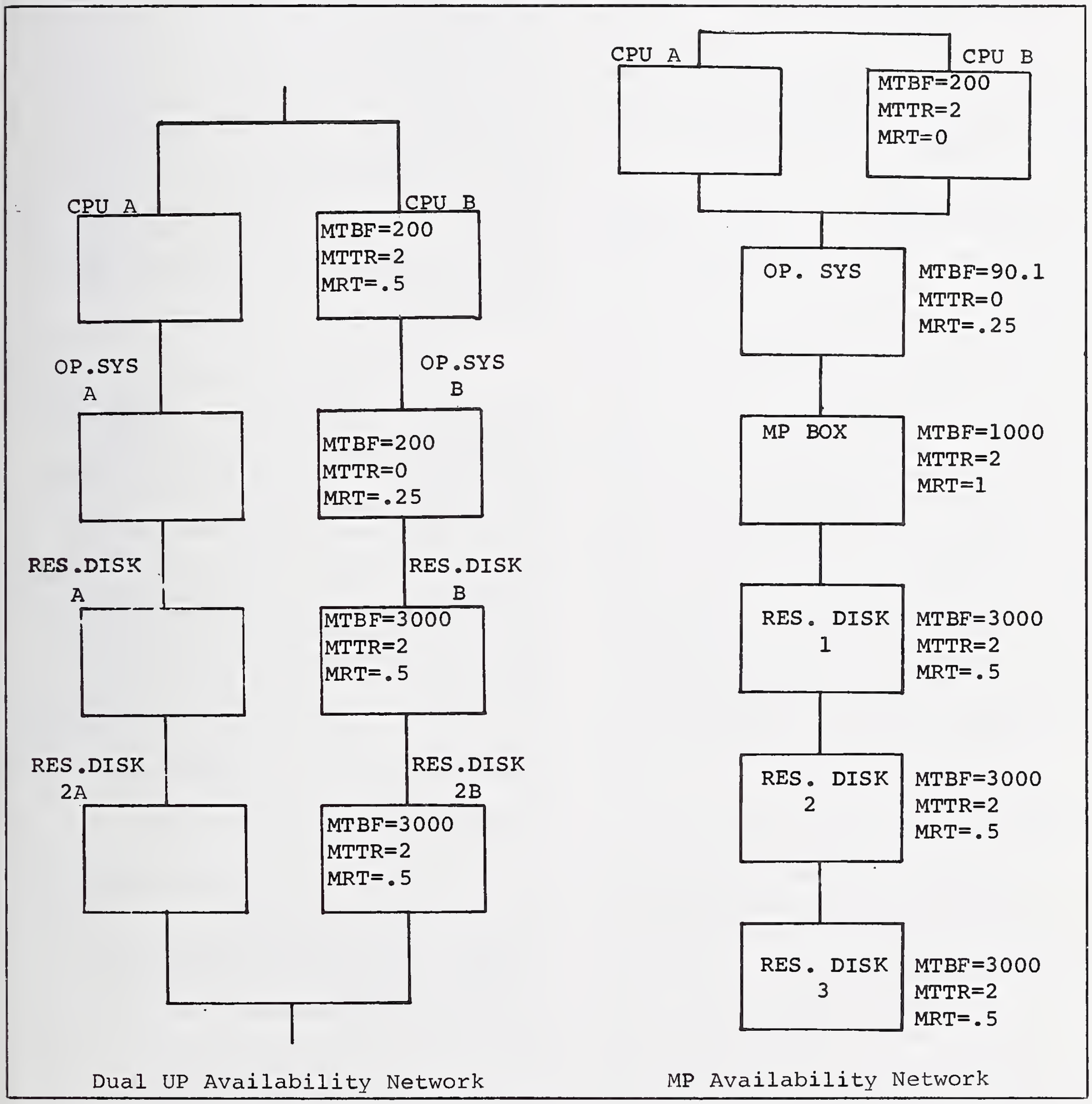


CHART I

CPU's

Each CPU is charcterized by a mean time between failures (MTBF) of 200 hours and mean time to repair (M TTR) of two hours. Thus, we would expect failures to occur as follows:

1. CPU (A) or CPU (B) or both - Once every 100 hours.

2. CPU (A) or CPU (B) but not both - Once every 101 hours.

3. CPU (A) and CPU (B) - Once every 10,000 hours.

Simultaneous CPU failure is not of interest because it has an identical impact on both environments. The above results were obtained by solving for the combined availability of 2 CPU's and then the mean time between simultaneous failures which was subtracted from the total failure rate of .01 per hour yielding a mean time between partial failures of 101 hours. Division of 4000 operational hours by a 101 hour MTBF indicates that we should expect 39.6 failures per year each affecting 50% of a dual UP facility. If no recovery were possible on the adjacent CPU, 79.2 hours per year would be lost. An average recovery time of 30 minutes, which is more realistic, yields 19.8 annual lost hours that would not occur on an MP. We would save $19,800.00 per year if we had a MP.

Operating System

Operating System software failures have a tendency to offset MP benefits. To illustrate, we have divided them into four categories:

1. Stopped both elements of a MP

2. Stopped one element of a MP

3. Stopped one UP

4. Stopped both UP's

Categories 1 and 3 deserve serious consideration, 2 and 4 are highly improbable, although type 4 failures occasionally result from bungled systems programming efforts. Another important point is that software failure rates are tightly correlated to workload level, given software of identical quality and complexity. This means that MP software should be expected to fail at least as frequently as the combined rate of two UP's. As a matter of fact, MP rates could easily be a little higher because of increased environmental complexity. Note, however, that when MP software fails, the entire site is out, not just half the site as in a UP software failure. If the mean time between UP software failures is 200 hours then we can expect the MP to fail once every 100 hours.

Simultaneous independent UP software failures are extremely improbable. Assuming 15 minutes to recover from an individual CPU software problem, we can expect to lose 2000 terminal hours on the MP and 1000 on the two UP's. This occurs because each UP failure brings down only 50% of the terminals. Our loss would be $10,000.00 Dollars per year. To account for increased complexity, we should add 10% to the MP failure rate, these extra failures would not occur in a UP. Thus we have an additional $2000.00 per year in software offsets to hardware availability benefits.

MP Interconnection Box

Interconnection Box failures are not possible in a dual UP environment because such a component is not part of that configuration. They would also tend to offset the MP's CPU availability benefit.

We can assume that the MP box is very reliable, it is probably 20% as complex as the CPU so its mean time between failures should be approximately 1000 hours. We can thus expect 4 failures per year during on-line operation. At worst, the MP would have to be split in such circumstances, a process again requiring 30 minutes for software recovery. In this case, however, the whole site would be inoperable, resulting in 400 lost terminal hours or $4000.00 per year.

System Support Disks

Failure of this component will stop both halves of an MP but only 50% of a UP. Our configuration show 2 disks for each UP and a total of 3 on the MP. Each UP will experience a disk failure once every 1500 hours with an MRT of 30 minutes resulting in an annual loss of $5333.00. The MP will experience a failure of all terminals once every thousand hours resulting in an annual loss of $4000.00. Thus, we have a net gain in the disk area of $1333.00 per year.

Net Availability Impact

As a whole, availability has the following financial impact:

1.	CPU	Saves $19,800.00
2.	Operating System	Costs $12,000.00
3.	MP Interconnection Box	Costs $ 4,000.00
4.	System Support Disks	Saves $ 1,333.00
	NET ANNUAL SAVINGS	$ 5,133.00

Thruput Details

Two factors are at work, in opposite directions, influencing thruput. Multiprocessor configurations have a tendency towards higher thruput because CPU vs input/output segmentation is reduced. In other words, MP's reduce the probability of a total I/O wait on CPU A while two programs are ready for CPU B. In opposition to this, is the coupling loss resulting from two machines in an MP contending for memory accesses to table entries or instruction locations. These losses range from averages of 10 to 25%. Our analysis indicates that both factors, reduced segmentation and coupling losses, are sensitive to CPU loading in opposite directions. When CPU load rises the probability of a segmentation problem in a pair of UP's is reduced and, the probability of memory access conflict increases in a MP.

Thus, we feel that MP thruput is theoretically superior at low utilization and UP thruput is superior at high utilization. Unless we are faced with a utilization reduction caused by CPU vs. I/O segmentation, additional thruput at low utilization (which now must be a result of low demand) is not meaningful.

To illustrate these results, we have prepared Chart II, which is slightly optimistic about MP thruput at high utilization and pessimistic at low use levels. Table entries compare thruput of an MP to that of a pair of UP's at various levels of UP utilization and MP coupling loss factors. Table entry values were computed by solving a queueing algorithm based on a fully cooperative service facility serving a finite number of customers for actual use of the central server. CPU utilization levels were converted to thruput and the thruput of a MP was divided by that of two UP's. Average elapsed time per job in the UP at a particular utilization level was held constant for the thruput comparison at each coupling loss level. Our curves are not smooth as we go across utilization levels because average elapsed time could not be held perfectly constant in the queueing algorithm which works with integer mix sizes. It can be seen that at 70% utilization, a likely operating point, we can expect a slight decrease in the thruput of an MP compared to two UP's. If our shop had an average daily utilization of 70% we would expect to see quite a few moderately high volume days with average utilizations of 80% and a few at 85 to 90%.

On those high volume days we would discover that the shop's thruput is a little lower than we computed it to be, based on data taken from low and average volume days. Perhaps we have discovered some relativistic effects which should be taken into account by software physicists.

If, on the other hand, we are dealing with low utilization resulting from CPU vs. I/O segmentation, the system should be tuned prior to studying the MP problem. After all non-CPU constraints are released, thruput will be limited by available workload or very stringent elapsed time requirements. Objective analysis of MP vs. UP thruput can now be performed as described above.

Cost Analysis

Implementation of an MP will impact the following types of cost:

1. CPU operator salaries
2. Systems programming salaries
3. Maintenance costs
4. Terminal operator costs
5. Equipment cost

To determine operations staff changes that would result from implementing an MP we consult the technological matrix that relates our site staffing requirement to workload and enviromental complexity. Our inputs are one fewer CPU's, a more complex environment, and everything else the same as before. The output would probably be a recommendation to reduce staffing by one console operator on each shift and to add a Systems Programmer to the first shift. Operator salaries, including benefits range from $20,000.00 to $25,000.00 per year. Systems programmers earn, including benefits, $27,000.00 to $36,000.00 per year. These salaries should be affected by inflation to different degrees with a somewhat larger effect expected for Systems programmers. This minor reduction in personnel occurs because personnel is to a much larger extent a function of workload than CPU population.

Additional maintenance will cost approximately $10,000.00 per year and will inflate at a rate equal to that of System Programmers. Terminal operator cost reductions were calculated to be approximately $5133.00 per year in a 200 terminal environment. These will inflate at the same rate as that expected for console operator salaries.

An MP requires core storage for one, instead of two operating systems and reduces disk storage necessary to contain the operating system. We estimate that$12,000 Dollars per year can be saved in disk area. Reduced core utilization would result in a $15,000 per year savings. In total these hardware reductions would account for a $2700.00 per year maintenance reduction.

From a costing viewpoint, we are being asked whether a capital investment in an MP box would have a sufficiently high return to be warranted. An initial investment of $400,000.00 is required and a reasonable internal rate of return would be 12%. We can expect a series of cash flows in future years as shown in Chart III.

The present discounted value of the savings resulting from implementing a MP would be $243,380.00, clearly to low to justify spending $400,000.00. These results are relatively insensitive to terminal operator population because each additional 200 operators increase the return by only $5000,00 in year one. It appears that more research in the area of additional benefits will be required before this MP can be rationally justified in an average commercial environment. However, one should not underestimate the ability of Industrial Engineers to uncover more hidden costs associated with MP operation.

CHART II

MP/UP THRUPUT RATIOS FOR EQUIVALENT TIMELINESS FACTORS

MP COUPLING LOSS FACTOR	UNIPROCESSOR UTILIZATION LEVELS						
	50%	60%	70%	75%	80%	85%	90%
1.6/2	.92	.87	.89	.89	.89	.86	.84
1.7/2	1.05	.98	.97	.96	.98	.91	.89
1.8/2	1.17	1.03	1.04	1.03	1	.98	.95

CHART III

ANNUAL CASH FLOW

Cost Area	Year 1 DF=.9448	Year 2 DF=.8435	Year 3 DF=.753	Year 4 DF=.672	Year 5 DF=.60	Infla-tion Factor
CPU Operator	+65672.	+62172.	+58855.	+55697.	+52734.	.065
Sys. Programmer	-30858	-29617.	-28424.	-27271.	-26177	.075
Maintenance	-9796	-9402.	-9024.	-8657.	-8310.	.075
Maintenance	+2645	+2539.	+2436.	+2337.	+2244.	.075
Term. Operator	+4994.	+4727.	+4476.	+4236.	+4012.	.065
Disk and Core	+25510.	+22775.	+20331.	+18144.	+16200.	0
Annual Savings	58167.	53194.	48350.	42966.	40703.	

CPEUG 78

COMPUTER PERFORMANCE MANAGEMENT

COMPUTER PERFORMANCE MANAGEMENT

Philip J. Kiviat

SEI Computer Services
Washington, D.C.

CPE has only recently emerged as a full-fledged technical/management activity in most computer organizations. With a history going back scarcely more than ten years, CPE has more ahead of it than behind it in the way of tradition, standard practices or discipline.

In keeping with its emergence as a necessary computer management activity, CPE is filling in the gaps in its architecture. Formerly preoccupied with "tools" such as hardware monitors, CPE practitioners are now spending their time understanding what needs to be done, and how that might be done, rather than learning the fine points of particular measurement techniques. There is a shift from the specific to the general as more systematic approaches are being developed and applied to the analysis of computer systems.

In keeping with the trend, this session contains three papers that extend and codify the practice of CPE. Two papers provide a formal structure for two core CPE tasks - system tuning and system sizing. They are both the product of work done at the Federal Computer Performance Evaluation and Simulation Center for Federal agencies that felt the need to formalize preferred approaches to these tasks. A third paper extends the measurement process into the realm of human performance, and shows how human performance variables can be incorporated along with conventional hardware/software measurements as part of the systems procurement and systems evaluation phases of Federal ADP programs.

This session thus both extends the realm of CPE into areas that concern themselves with human behavior, and narrows it by presenting structured approaches to CPE tasks that are now done with a great deal of individualism in many places.

THE DEVELOPMENT OF A TUNING GUIDE

Barry M. Wallack

Command and Control Technical Center
Computer Performance Evaluation Office
The Pentagon Washington, DC 20301

The Federal Computer Performance Evaluation and Simulation Center under contract to the Command and Control Technical Center has developed a document for Worldwide Military Command and Control Systems that can be used by site personnel to analyze the performance characteristics of their Honeywell 6000 computer systems. This document, called an H-6000 Tuning Guide, incorporates detailed analysis procedures that guide the analyst in applying specific techniques to improve system performance.

Key words: Computer; Honeywell 6000; performance evaluation; response time; tuning; turnaround time; WWMCCS.

1. Introduction

The Office of the Joint Chiefs of Staff (JCS) has directed that the Command and Control Technical Center (CCTC) develop a computer performance analysis capability to support the World Wide Military Command and Control System (WWMCCS).

CCTC, acting at the direction of the JCS, has specified that WWMCCS ADP managers are to apply various computer performance evaluation (CPE) tools and techniques to the systems now running at their sites. CCTC has also defined the need to instruct WWMCCS technical personnel in the selection and application of the CPE tools and techniques appropriate to individual WWMCCS ADP sites.

CCTC asked FEDSIM to plan and implement a document that could be employed by WWMCCS ADP personnel to diagnose problems and propose changes that would improve the performance of WWMCCS ADP systems.

The objective of the resulting FEDSIM project was to provide all WWMCCS installations with a document that could be used by staff personnel to analyze the performance characteristics of their ADP systems. This document, called an <u>H-6000 Tuning Guide</u>, was to contain sets of analysis procedures to improve system performance. The Guide presents a precisely structured system of procedures for the analysis of WWMCCS computer services and systems. While the Guide is precisely structured to the H-6000 and employs tools directly available only to WWMCCS sites, the importance of this project lies not in the procedures themselves, but in the concept and structure of the Guide. For the first time, the techniques and methodology of performance evaluation (i.e. tuning) are described in a detailed, step-by-step, cook-book document. Performance evaluation is no longer a black-magic art. There is a logical set of steps and procedures that can be followed when analyzing a computer system. It is no longer a matter of collecting reams of data, and then not knowing what to do with that data. There is a structure to the thing called Performance Evaluation. It is a science not an art.

While the Tuning Guide accomplishes the above for the H-6000 it is the hope of CCTC that this idea will spread to other vendors and that Performance Evaluation will enter the world of science and leave the world of Black Magic.

2. Scope of Computer Performance Tuning

In one way or another, the performance of a computer system is influenced by nearly every facet of the data processing function.

The following examples illustrate the scope of the computer performance tuning process.

2.1 System Design

Computer application system design and development can be the starting point for performance degradation. Errors in original design with respect to I/O media selection, file structures, frequency of run, etc. may result in less than optimal performance for as long as an application is in existence.

2.2 Programming

A programmer's proficiency and the availability of program optimization tools, for example, will influence program design and coding, and affect system performance.

2.3 Hardware Configuration

Specific components of a computer system may be mismatched to the system as a whole, causing major subsystems (or the entire configuration) to operate at a reduced performance level. Even if the performance capabilities of the individual subsystems are reasonably well matched, the system may be poorly configured for the site's workload, resulting in poor performance.

2.4 System Software

The software supplied by the mainframe vendor may be inappropriately parametrized to fit the site workload, or may be a source of high overhead or bottlenecks to efficient workload processing.

2.5 Operations

An operations staff schedules the workload, provides job assembly (and library) services, and operates the system through the console. All of these functions are vital to the proper operation of the system. Mistakes, insufficient training, poor documentation, and a variety of other reasons may contribute to operational problems which substantially decrease system performance.

2.6 Communications Hardware and Software

A communications network, its interface to a central system, and the software used to control the on-line applications may have a significant impact on the system's overall performance.

3. Tuning Procedure

The process of analyzing and appropriately adjusting computer system performance variables is known as computer system performance tuning. The following termonology is used throughout the guide.

3.1 WWMCCS Services

Within the context of the Guide, WWMCCS system services are: (1) batch job services and (2) GCOS Time Sharing System services. The service time for batch workload service is called batch turnaround time and the service time for TSS service is called TSS response time.

3.2 Turnaround Time

This is the total elapsed time taken by a job (or set of jobs) submitted to a WWMCCS site for batch processing. Batch turnaround time is comprised of computer system processing and physical input and output handling in the machine room, both before and after system processing. A job's turnaround time therefore includes all processing and waiting points through which the job must pass from submission until return to a user. The Guide's structured batch turnaround time analysis examines these processes and waiting points.

3.3 TSS Response Time

A time-related structure similar to batch processing turnaround time can be conceived for TSS service. The Guide's TSS response time analysis defines TSS Response Time as the elapsed time between an on-line user's request for service from the system and the system's request for further input from the user. The TSS analysis procedures of the Tuning Guide devide response time into processes and waiting points associated with: (1) CPU service, (2) disk I/O, (3) memory, (4) terminal I/O, and (5) special system processes.

3.4 Service/Resource Link

Guide analysis procedures use measures of system resources to analyze performance degradation. The Guide uses these resource measures (1) to isolate the processes or queue points that are major contributors to a particular elapsed service time, (2) to hypothesize causes of the elongation of these processes, and (3) to test the validity of the hypotheses, confirming the source of performance degradation. For example, the batch turnaround time analysis directs investigation to the

GCOS process that is exhibiting the longest elapsed time of all GCOS system processes. This process (e.g., Core Allocation) require system resources (i.e., core) to perform its service. A lack of these resources elongates the service (i.e., core allocation) to batch jobs.

3.5 Hypothesis Confirmation

The tests that the analyst is directed to conduct are used to confirm specific hypotheses as causes of the service elongation. These confirmations involve examining specific system software or performance data reports.

Particular resource bottlenecks may be confirmed as elongating turnaround or response time. The Guide's analytical structure proposes specific solutions to correct or improve the degraded system performance. Several solutions can usually be applied to remove a particular bottleneck. In general, the Guide procedures provide up to four kinds of solutions to remove identified bottlenecks:

a. Scheduling Solutions. These solutions change the way that either batch or TSS workloads are scheduled for processing. They shift particular workloads to more evenly distribute system resources across the workload.

b. Parameter Solutions. These changes involve adjustments to system or subsystem functions. Examples include: (1) changes to the parameters of the GCOS Dispatcher or (2) a change in the placement of GCOS libraries. A solution may include specific changes to GCOS code, made through authorized software patch procedures.

c. Programming Solutions. These changes can involve modification of one or more application jobs running in the system. For example, Guide procedures are provided to investigate a program's execution characteristics in order to determine where it spends most of its execution time. This assists the programmer in examining the code. At a simpler level, Guide recommendations are made to speed application jobs by changing particular file locations discovered as delaying the job.

d. Sizing Solutions. These types of system change involve an increase (or a decrease) in the system's hardware configuration.

The Guide solutions are presented in a sequence that makes them easiest or least expensive to implement. Guide procedures direct the analyst to solutions that involve additions of new equipment only after other techniques have been tried.

Figure 1 is a flow chart of the analysis process used throughout the Guide. The analysis process is comprised of two phases: (1) a Problem Definition Phase and (2) a Problem Analysis Phase. The activities of the Problem Definition Phase are directed toward determining whether a batch turnaround time or TSS response time problem actually exists. The activities of the Problem Analysis Phase are directed toward revealing causes of the identified turnaround time or response time problem.

The Guide batch turnaround time model (see Figure 2) is a conceptualization of the WWMCCS system components of batch turnaround time. The model defines the processes and phases through which batch jobs pass as they are being processed by a WWMCCS system. Note from Figure 2 that the batch turnaround time model uses a three-level structure to assist in the search for batch turnaround time bottlenecks. Jobs are classed as Local Batch, Remote Batch "A", or Remote Batch "B", depending on their source and the type of output they produce. Batch turnaround time is divided into three phases: Pre-Processing (before the job is entered into the WWMCCS system), System Processing, and Post Processing (after the WWMCCS system has finished the job). Each phase is divided into processes. System Processing is divided into the seven processes shown in Figure 2. Pre-Processing and Post Processing vary from site to site and their processes must be defined locally.

The TSS response time model is similar to the batch turnaround time model in that it divides elapsed time for a user request into several different categories. The TSS response time model divides the time spent by a user at a terminal into waits and services associated with CPU Time, Disk I/O Time, Memory Wait Time, and Special Waits. The model further subdivides each of these categories into two or more subcategories. The amount of response time associated with each category and subcategory guides further investigation into improving response time. Again, choice of a particular category for further investigation depends on the amount of response time associated with that category.

Figure 3 provides a graphic description of the TSS Response Time Model.

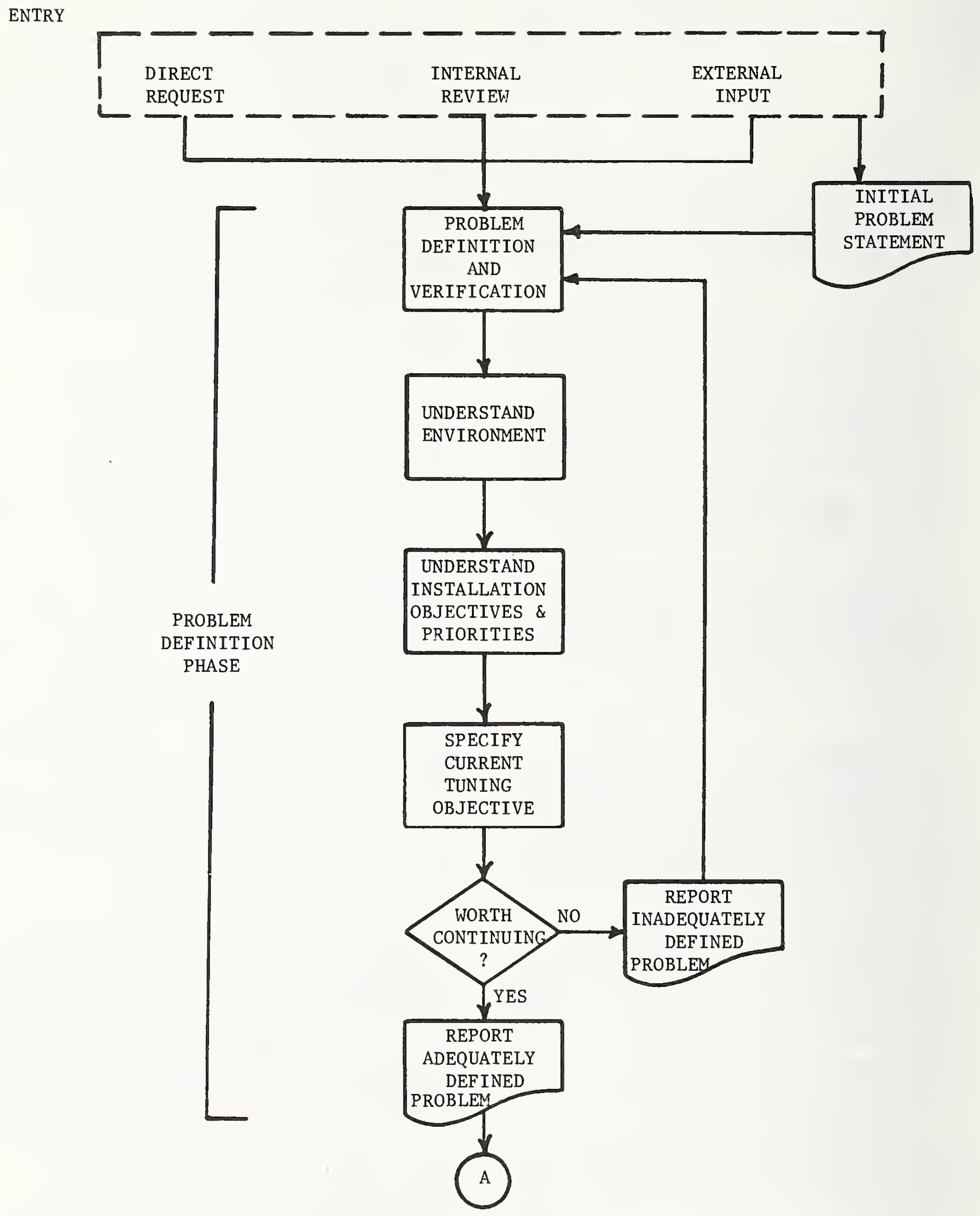

ENTRY
DIRECT REQUEST
INTERNAL REVIEW
EXTERNAL INPUT
INITIAL PROBLEM STATEMENT
PROBLEM DEFINITION AND VERIFICATION
UNDERSTAND ENVIRONMENT
UNDERSTAND INSTALLATION OBJECTIVES & PRIORITIES
SPECIFY CURRENT TUNING OBJECTIVE
WORTH CONTINUING ?
NO
REPORT INADEQUATELY DEFINED PROBLEM
YES
REPORT ADEQUATELY DEFINED PROBLEM
A
PROBLEM DEFINITION PHASE

Figure 1. Flow Chart of Tuning Process

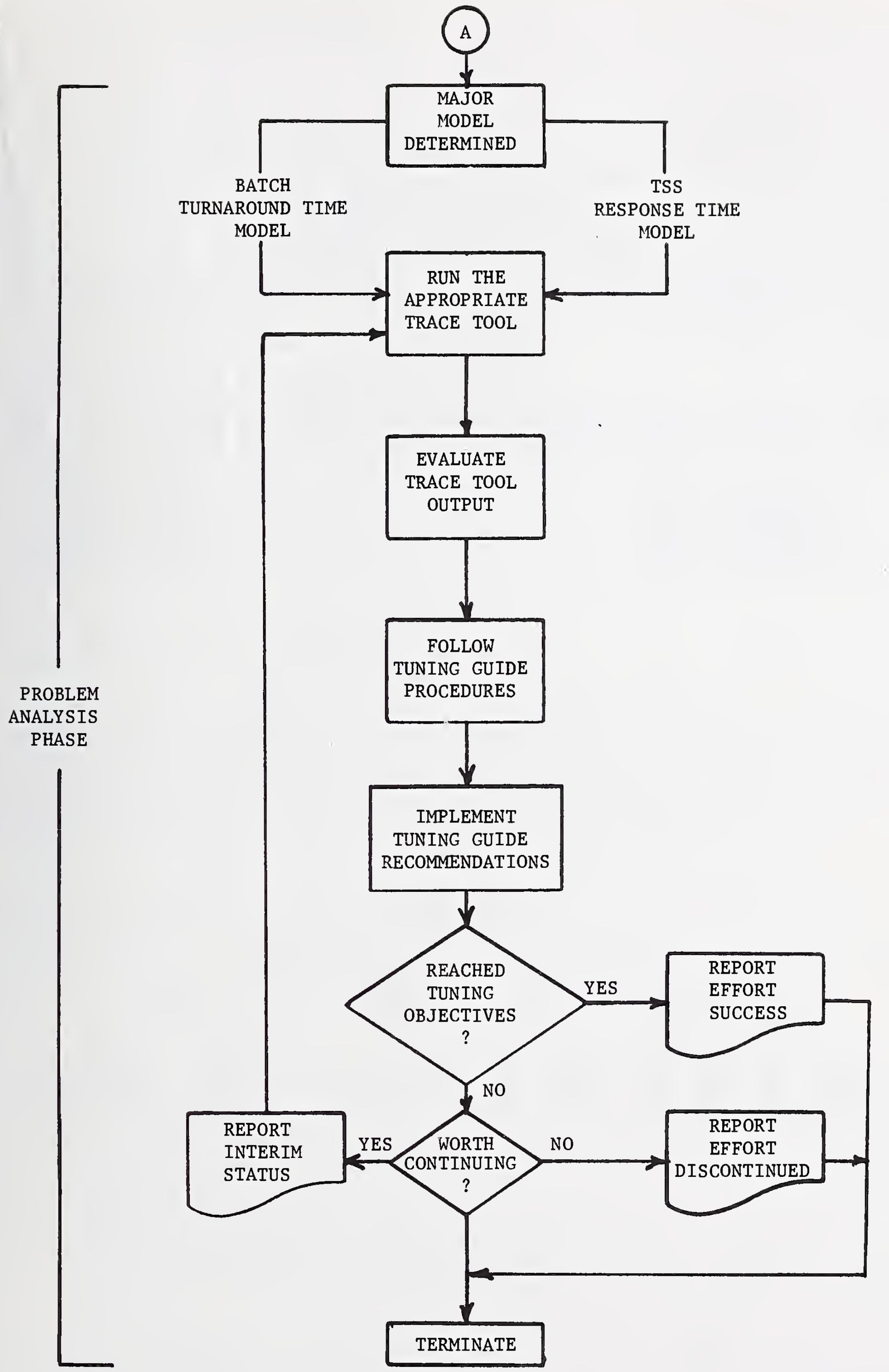

A
MAJOR MODEL DETERMINED
BATCH TURNAROUND TIME MODEL
TSS RESPONSE TIME MODEL
RUN THE APPROPRIATE TRACE TOOL
EVALUATE TRACE TOOL OUTPUT
FOLLOW TUNING GUIDE PROCEDURES
IMPLEMENT TUNING GUIDE RECOMMENDATIONS
REACHED TUNING OBJECTIVES ?
YES
REPORT EFFORT SUCCESS
NO
WORTH CONTINUING ?
YES
REPORT INTERIM STATUS
NO
REPORT EFFORT DISCONTINUED
TERMINATE
PROBLEM ANALYSIS PHASE

Figure 1. Flow Chart of Tuning Process (Cont'd)

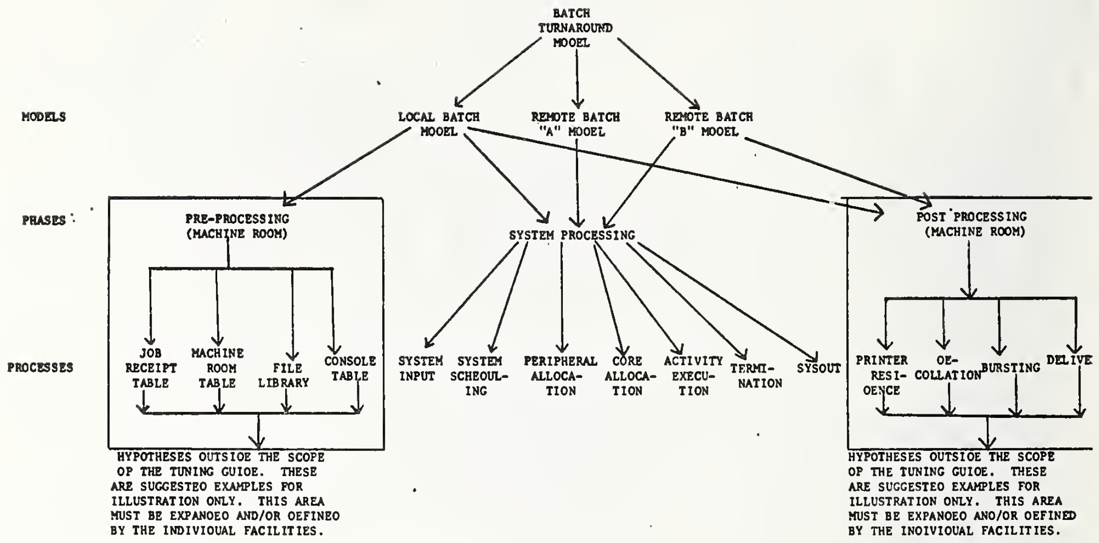

BATCH TURNAROUND MOOEL
MODELS
LOCAL BATCH MOOEL
REMOTE BATCH "A" MOOEL
REMOTE BATCH "B" MOOEL
PHASES
PRE-PROCESSING (MACHINE ROOM)
SYSTEM PROCESSING
POST PROCESSING (MACHINE ROOM)
PROCESSES
JOB RECEIPT TABLE
MACHINE ROOM TABLE
FILE LIBRARY
CONSOLE TABLE
SYSTEM INPUT
SYSTEM SCHEOUL-ING
PERIPHERAL ALLOCA-TION
CORE ALLOCA-TION
ACTIVITY EXECU-TION
TERMI-NATION
SYSOUT
PRINTER RESI-OENCE
OE-COLLATION
BURSTING
DELIVE
HYPOTHESES OUTSIOE THE SCOPE OF THE TUNING GUIOE. THESE ARE SUGGESTEO EXAMPLES FOR ILLUSTRATION ONLY. THIS AREA MUST BE EXPANOEO AND/OR OEFINEO BY THE INDIVIOUAL FACILITIES.
HYPOTHESES OUTSIOE THE SCOPE OF THE TUNING GUIOE. THESE ARE SUGGESTEO EXAMPLES FOR ILLUSTRATION ONLY. THIS AREA MUST BE EXPANOEO ANO/OR OEFINED BY THE INOIVIOUAL FACILITIES.

Figure 2. Summary of Batch Turnaround Time Model Structure

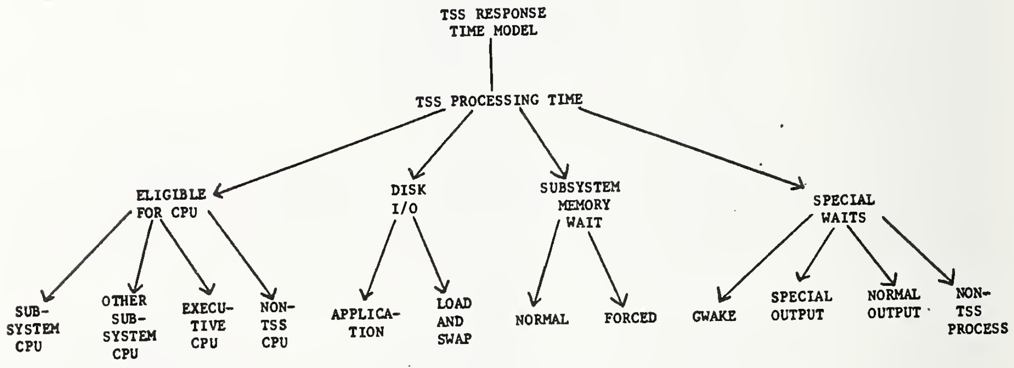


Figure 3. Summary of TSS Response Time Model Structure

The two Guide models employ special software to determine where a user spends the largest amount of elapsed time in their respective processes.

The Tuning Guide then provides instructions for evaluating initial analyzer outputs. These instructions guide the analyst to specific test procedures. For both analyses, the instructions are implemented through a form that is filled in by the analyst. This form directs the type and sequence of tests to be conducted during the search.

A Guide test procedure contains a series of analytical steps that are directed toward a diagnostic objective. For example, one of the procedures under the CPU Execution Characteristics Test in the batch turnaround time analysis has the following objective: "... to determine the dominant CPU user." The steps of these procedures involve: (1) examining specific reports to obtain performance data, (2) entering selected metric values on a form, (3) calculating ratios or percentages from these entry values, and (4) making certain tuning decisions (and subsequent recommendations) from the calculated values.

In some Guide test procedures, several CPE tools may be used to gather data. If this is the case, it is the analyst's responsibility to provide synchronized data (i.e., reports produced from multiple sources that are measuring the systems over a common elapsed time).

Each test procedure uses a form on which the significant measurement values are entered. The form helps clarify the procedures; use of the form documents the tuning effort for later management and analyst reference.

A ratio analysis technique is used to compute values which are quantities compared with tuning decision criteria. As an example, one of the procedure tests results in the calculation of an Activity CPU Ratio and a GCOS CPU Ratio. These ratios are then employed in a decision step to further direct the test. All computations required in each test are made directly on the appropriate test form under the direction of a procedure step.

The analyst is directed during or at the end of each test to a series of tuning recommendations. These recommendations incorporate generally accepted system tuning practices as applied directly to the WWMCCS system.

Analysts should determine whether tuning objectives have been met after each implementation of a tuning step. Even if a tuning objective is not met at any particular iteration of tests and tuning step, an analyst may observe an improvement in turnaround time or TSS response time. This improvement will act as a checkpoint, indicating that the overall analysis is proceeding in the right direction. If it appears difficult to reach a tuning objective after several attempts, it is possible that the original tuning objective may have been over stated. In this case the analyst may want to reformulate the tuning objectives.

This paper has briefly described the structure and philosophy of the H-6000 Tuning Guide. The specific tests and a detailed description of the evaluation tools has been left out so as not to restrict this technique to the H-6000. It is believed that this approach is applicable to all computer systems and could result in a greater understanding and appreciation of what Performance Evaluation really is.

The development of this guide required many long hours of work and dedication from FEDSIM personnel, especially Dr. John Peterson and Mr. Hal Stout. Sections of this paper have been extracted from the H-6000 Tuning Guide.

GUIDANCE FOR SIZING ADP SYSTEMS (ADPS'S)

Dennis M. Gilbert
James O. Mulford
Mitchell G. Spiegel

Federal Computer Performance
Evaluation and Simulation Center
Washington, DC 20330

This paper provides a structured methodology to assist a sizing team in making a thorough definition and analysis of new requirements, ADPS alternative selections, and workload impact. Suggested sizing tools and techniques are presented, and guidance is included to aid a sizing team in obtaining accurate and timely results. While this paper is not a complete text on sizing, it can be used as a basic guide for requirements identification, feasibility, and impact studies, and should be enhanced with a sizing team's expertise, consultation from sources with sizing experience, and reference to other sizing literature.

1. Introduction

Numerous Federal Regulations establish a comprehensive method for managing ADPS and ADPS elements from the initial statement of requirements to completion of the ADPS life cycle.[1] Such documentation as Requirements, Project Plans, Request For Proposals (RFP's), and ADPS Specifications and Test Plans, must be provided at key decision points in the ADPS life cycle (e.g., concept certification, requirements review, design certification, solicitation, testing, and final operational review). Documentation is essential in evaluating, planning, programming, and budgeting new requirements. The information contained in these documents must be thorough, accurate, and timely.

Sizing studies can greatly assist analysts who identify, gather, analyze, and interpret information to support preparation of the above documents. Sizing is a process for (1) analyzing requirements, specifying alternative solutions, evaluating workloads, and (2) estimating the amounts and location of people, hardware, software, and communications resources necessary to process that workload within predefined operational constraints. It combines (1) a sound methodology, (2) proven tools and techniques, and (3) functional and ADP expertise, in projecting ADPS needs.

The paper is directed toward those levels of management and technical staff that are involved in an ADPS life cycle

[1]Federal Management Circular 74-5, "Management, Acquisition and Utilization of ADP"; Office of Management and Budget (OMB) Circular A-11, "Preparation and Submission of Annual Budget Estimates"; OMB Circular A-109, "Major System Acquisition"; Federal Property Management Regulation (FPMR) 101-32.11, "ADP Communications Support for ADP Systems"; FPMR 101-32.15, "Future Plans for ADP and Telecommunications Systems"; Federal Procurement Regulation (FPR) Subpart 1-4.11, "Government-Wide Automated Data Processing Equipment, Software, Maintenance Services and Supplies"; and Air Force Regulation 300-12, Volume I.

study[2] and the application of its results. Sections 1 and 2 are directed toward upper and middle management. Sections 3 and 4 are written for those personnel who either conduct, review, or use sizing studies, ADPS and user management. Section 5 provides a sizing team with references for methodology and guidance. The objective of this paper is to guide a sizing team toward accurate, timely, and thorough information in sizing studies, through consistent application of a sizing methodology.

The reader must keep two basic principles in mind while reading and using the described sizing methodology and guidance. One is that the methodology and guidance are descriptions of good practices for most sizing studies. They do not cover, nor are they applicable in, all situations. The second is that the methodology and guidance stress reasonableness in all practices and procedures. The user is responsible for determining the exact sizing approach taken, conducting the sizing study, and extrapolating the results to support the requirement. Any question of sizing procedure or technique should be evaluated in this context.

The sizing methodology steps usually cannot be followed as a "recipe" with successful results. Instead, this report represents good practices associated with areas of concern. In this sense, the paper is useful as a checklist and, to some degree, identifies areas where special competence, expertise, or particular attention may be required.

2. Sizing Methodology Overview

2.1 Sizing and the Computer System Life Cycle

Personnel involved in decisions affecting a computer system must be aware that sizing studies can support decision making throughout a computer system's life cycle. Table 2.1[3] illustrates seven phases in an ADPS life cycle. In each phase, sizing studies can provide information required to make effective and efficient decisions. Because of this, a sizing study conducted for a new requirement should not only satisfy the sizing objective (primary) but also provide a foundation for future studies in the ADPS life cycle (secondary).

The following elements are desired in a sizing study:

(1) A clear statement of a study's objective(s), including time and funds available for a study

(2) A methodology to characterize and analyze the workload, design sizing experiments, identify sizing alternatives, conduct sizing experiments, analyze findings, and present sizing results

(3) Calibrated tools and techniques which can provide information (in appropriate units of measure) needed to satisfy the sizing objective

(4) A team of knowledgeable functional and ADP analysts

(5) Assistance from (a) centers with sizing expertise and (b) available sizing literature

(6) Management interest and participation

2.2 Management's Role

For a sizing study to provide good decision-making information, full participation by ADPS, user, middle and upper management is required. At a minimum, management must select the sizing team members, provide guidance for designing the sizing study, allocate funds and allot time to complete a study, guide and track a study's progress toward a successful satisfaction of the sizing objective, and understand the uncertainties associated with the use of sizing study results.

2.2.1 Selecting the Team

The sizing team leader should be an ADP/Communication/Software specialist (as opposed to a functional specialist), since it is Data Processing's responsibility to select the most cost effective ADPS alternative(s) that will meet the user requirements. Ideally, the team is composed of (1) functional analysts familiar with the user requirements to be automated, (2) sizing specialists (in-house, contractors, or others) familiar with sizing methodologies, tools, and techniques, and (3) systems analysts familiar with both user requirements and the organization's ADP objectives. Depending upon the scope of a

[2]Portions paraphrased from "Guidelines for Benchmarking ADP Systems in the Competitive Procurement Environment," FIPS PUB 42-1 (Washington, DC: May 1977), p. 5.

[3]"Guideline on Computer Performance Management: An Introduction," FIPS PUB 49 (Washington, DC: May 1977), p. 9.

Table 2.1. Sizing Study Support in Computer System Life Cycle

ADP System Life Cycle Phase	Possible Uses of Sizing
(1) Requirements Analysis	Provide Information for Feasibility Study
(2) General System Design	Evaluate Alternative Organization, Hardware, Software, and Communications Architectures
(3) Requirements Specification and Request for Proposals (RFP)	Provide Performance and Workload Information for Specifications and RFP
(4) Vendor Evaluation and Selection	Assist in Evaluating Proposals' Performance
(5) System Delivery, Installation, Acceptance, Detailed Design	Identify Required Configuration of ADPS
(6) System Operation	Identify Potential Bottlenecks in Future Processing
(7) System Enhancements	Predict Performance of System for Enhancement Alternatives

study, the team size may range from two to several members. Initially, a small group (two to five people) should be identified to assist management in defining the sizing objective and the effort required to complete a study. Based upon this group's recommendation, management can then assign additional personnel to the study.

2.2.2 Designing the Study

The initial group of analysts selected for a study is tasked with defining its scope. Objectives, constraints, assumptions, an initial list of alternatives, information needed, and known sources of data are included in the sizing study definition. During this task, management frequently interacts with the team members. Once the scope of the study is determined, it is management's responsibility to review the definition and ensure that it will satisfy their information needs. During this review, management should check that the definition does not constrain or limit the sizing alternatives. The sizing study must be oriented toward alternative analysis and not toward the justification of a specific alternative.

2.2.3 Allocating Funds and Allotting Time

Management must carefully review a study's objective and scope to determine the estimated funding and required time for the study's completion. This is a very crucial part of a study that requires analysis of a study's benefits versus the required time and cost.

2.2.4 Guiding and Tracking the Study's Progress

As in any study, it is management's responsibility to carefully track a sizing study's progress. Figure 2.1 outlines the steps of a sizing methodology and key management review points. Management should guide a study toward satisfaction of the sizing study objectives, and not toward the selection of a particular alternative. In order to provide quantitative data for decision making, a study must be as free of subjective considerations as possible.

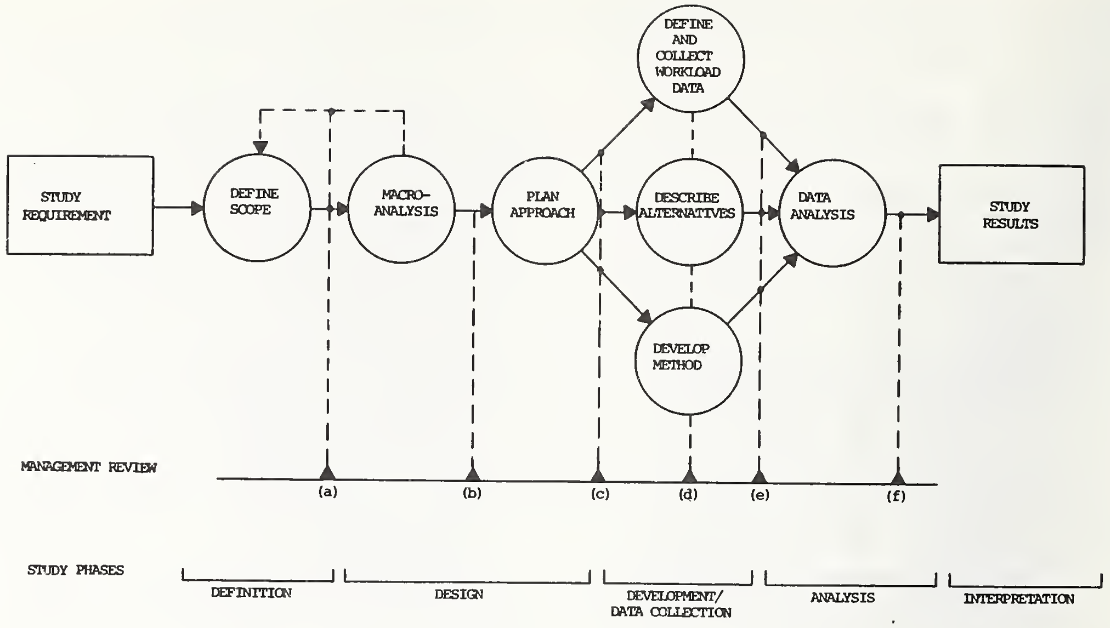


FIGURE 2.1. SIZING STUDY METHODOLOGY

2.2.5 Using the Sizing Results

The results of a sizing study are ADPS size estimates with some degree of uncertainty. Uncertainty can enter into the results from assumptions and constraints, tools and techniques, data collection, workload projections, subjective interpretation, and team error. Most likely, all factors will affect the accuracy of the results. Because of this, users of the sizing results must understand what these uncertainties are and what impact the uncertainties have upon the accuracy of results. Management must insist that the impact of these uncertainties be identified in the final documentation.

3. Sizing Methodology and Guidance

3.1 Introduction

This section is subdivided into the logical steps of the problem solving methodology, so that any sizing study will roughly parallel this section's flow. The intent of this methodology is to guide sizing studies toward a thorough analysis and a timely and accurate assessment of ADPS alternatives. It can be supplemented by other sizing information and individual expertise.

Two very important principles apply to the use of this methodology. First, document assumptions, observations, procedures, and intermediate results as the study progresses. Good documentation significantly reduces the time required to report the results at a study's conclusion, and provides an excellent reference for the sizing team. Second, return to previous steps in the methodology and review all completed work if significant aspects of the sizing change. Where applicable, redo all work affected by the changes. Assumptions, constraints, or objectives must sometimes be modified as additional information becomes available. When this occurs, identify the impact of these changes in the sizing study. If time and funding permit, incorporate these changes into the study by redoing the appropriate steps. If this is not feasible, then, at a minimum, document the changes, the parts of the study the changes affect, and, if possible, the relative impact the changes have on the sizing results.

3.2 Define the Scope of the Sizing Study

3.2.1 Attributes of the Sizing Study Definition

It is essential that the scope of each sizing study be clearly defined. This definition must convey the study's objective, constraints, and desired results. For example, the statement, "Determine the best ADPS to automate the inventory function"

provides little information. A better study definition would be:

"The objective of the sizing study is to (1) provide an estimate of the ADP workload that will be imposed by the implementation of the requirement for an automated inventory and (2) evaluate alternative methods of processing that workload. The study will provide size and cost estimates for the requirement's feasibility study, and results must be available by December 31, 19XX.

The study will provide estimates of processing, I/O, storage, and unique resource requirements of the proposed workload and will then size those estimates to at least the following alternatives:

(a) Implement the proposed workload on the existing ADPS, upgrading the staff, hardware, software, and communications, as required, or ...

(b) Obtain a second ADPS for processing only the proposed workload, or ...

(d) Obtain a second ADPS, and implement one of several possible mixes of current and proposed workload.

Size estimates and their associated cost estimates will be provided for alternative analysis. In addition, confidence in the results will be quantitatively expressed and potential workload variance impacts will be provided. The Functional Description Document for an automated inventory and draft Requirement Document are the primary reference documents for workload and requirements definition."

The importance of a clearly stated sizing study definition, structured to convey the correct information to the appropriate personnel, cannot be overemphasized. First, the identification of why a study is required and what results are needed should be separately stated and highlighted (e.g., first paragraph of objective statement). Second, constraints and information relevant to the study should be summarized in the objective statement and detailed in documents readily available to the sizing team. (The definition of a sizing study is the appropriate point to present information that is currently available and relevant to the study.) Third, the definition should be stated using terminology that is understood by management, functional analysts, ADP systems analysts, and sizing team members. Finally, the definition should be structured such that the progress and success in fulfilling the objective can be measured. Time, cost, and manpower constraints, as well as a statement of the nature and accuracy of required sizing results, should be included for this purpose.

3.2.2 Sizing Study Definition Considerations

Any sizing study definition includes (1) planning, programming, and budgeting (PPB) and (2) system capability considerations. PPB considerations often dominate a sizing study in the early stages of an ADPS (through the design phase and release of acquisition documents), and system capability considerations are usually emphasized thereafter. Table 3.1 lists some typical considerations of a sizing study definition. The sizing team should insure that the definition contains at least a subset of these considerations, since their presence identifies known constraints and clarifies desired results of the study (e.g., Are response time constraints a factor?; Is the desired result an estimate of total ADPS cost, or a detailed description of the configuration?).

3.3 Perform Macro-Analysis

Available sizing data should be submitted to a macro-analysis prior to initiation of the detailed sizing study. This analysis should provide for an understanding and expansion of the sizing study definition. The thoroughness, accuracy, and timeliness of a sizing study are highly dependent upon the sizing team's treatment of the considerations listed.

3.3.1 Understand Sizing Objective

The foundation for the macro-analysis is a knowledge of what information must be provided by the sizing study and what accuracy in the sizing results must be obtained. Each sizing team member should state in his/her own words the information and accuracy needs.

3.3.2 Determine Sizing Alternatives

The initial list of sizing alternatives should be reviewed, modified, and expanded to develop a complete list of viable alternatives. Alternatives that appear to be technically feasible, but which are not viable due to other considerations, should be documented along with the reasons for elimination from the list of alternatives. Factors to consider in determining alternatives include: Existing ADPS long-range plans, functional-user requirements, policy (functional, ADP, procurement), facilities, organizational structure, manning, and management guidance.

Table 3.1. Sizing Study Considerations

Programming, Planning, Budgeting (PPB)	
1. Cost	4. Facilities
2. Implementation Phasing	5. Manpower
3. Software Conversion	6. Risk Assessment
System Capabilities	
1. Response Time	4. Utilization of System Components
2. Throughput	5. Configuration Characteristics
3. Growth Potential	6. Technical Feasibility

3.3.3 List Known Assumptions and Constraints

It is the sizing team's responsibility to identify the relevant assumptions and constraints for the study. Since a sizing study includes an estimation of future workloads it is normal to have several assumptions and constraints. An initial list of assumptions and constraints must exist at the completion of the Macro-analysis.

3.3.4 Identify Required Data and Sources of Information

Based on the sizing objective and sizing alternatives, an initial list of data elements required to conduct the sizing should be developed. For each data element, the list should include the element's use (describe workload, describe hardware characteristic of an alternative, etc.) and potential sources for collection. It is important to identify the major data elements during the macro-analysis, since it must be determined (1) if the data is available and (2) if tools/techniques can be found to collect the data. Further discussion on data elements is provided in Section 3.3.5 (Select Tools/Techniques), Section 3.6 (Define Workload To Be Sized), and Section 3.7 (Describe Sizing Alternatives).

3.3.5 Select Tools and Techniques

a. <u>Types of Tools</u>. The three major categories of tools required for a sizing study are (1) data collection, (2) data analysis, and (3) sizing. The data collection tools and techniques assist the sizing team in defining the workload, human factors, and alternative hardware, software, and communication characteristics. They include hardware monitors, software monitors, accounting packages, documentation, interviews, and questionnaires. Data analysis tools allow the team to screen the raw data and establish relationships among variables. They include statistical techniques, display tools, and data reduction methods. The sizing tools provide the team with an aid for translating workload requirements into sized ADPS alternatives. The two major types of sizing tools are abstract models [analytic, static, and simulation] and workload models [benchmarks (synthetic and actual)].

The following paragraphs provide an overview of the various tools and techniques available for sizing studies. The sizing team should consult available references for additional information.

b. <u>Data Collection Tools and Techniques</u>

(1) <u>Measurement Tools</u>. Detailed data on an existing ADPS can be

collected using hardware monitors, software monitors, communication line monitors, remote terminal emulators, and accounting packages. Examples of data available from these tools include system component utilizations, resource usage per application work unit (job, transaction, interaction, step, etc.), frequency of occurrence of work units, system overhead (resource usage by system software), and workload variances across time. These data can be collected for sizing studies to define existing workloads, current ADPS characteristics and performance (for upgrade alternatives), and trends or workload characteristics that can be used to predict future workloads.

From the macro-analysis, the sizing team has a good understanding of what, if any, data measurements will be necessary. This understanding, and consultation with people who have experience with the available measurement tools, leads the sizing team to a proper selection of measurement tools for the study. Specific guidance for this selection process is as follows:

(a) Check with other sources (operations, systems, and higher management personnel) to identify what required data might already have been collected. Measurement by the sizing team may not be necessary.

(b) Check with knowledgeable sources (vendor, operations, systems, higher management, and CPE personnel) to identify the strengths and weaknesses of available measurement tools. For each tool, areas to address include accessibility, difficulty of use, associated overhead, accuracy of collected data, flexibility (how much can the amount of data collected and level of detail be varied), and the amount of expertise that can be provided to the sizing team.

(c) For each measurement tool, evaluate the amount of data validation and reduction that will be necessary to obtain the required information for the study. Insure that the time required for measurement, data validation, and reduction will not delay study results.

(2) <u>Documentation</u>. The sizing team should obtain and study all documents that affect the sizing study. Often data required for a sizing study is already documented and considerable time can be saved if it is made available to the sizing team. Items that may already be documented include (a) existing and future workload characteristics; (b) human factors, hardware, software, and communication component characteristics; (c) operational performance constraints (response times, turnaround times, security, availability, reliability, mobility, etc.); (d) growth projections; (e) cost/ benefit analysis; (f) assumptions for the study; and (g) additional ADPS alternatives.

Table 3.2 lists potential sources of documents. The sizing team should contact or obtain these sources (as well as others identified by the sizing team) to identify and collect documentation that is applicable to the sizing study. Table 3.3 lists several types of documents that often contain information which the sizing team can use in the study. A library of selected documents should be created for the study.

(3) <u>Team-Generated Techniques</u>. Usually, data obtained from measurements and documents will not fully satisfy the study's data requirements. Other commonly used techniques are interviews and questionnaires. Both techniques provide a mechanism for collecting data from knowledgeable sources in various areas of the study (see Table 3.2 for a list of sources.) Interviews establish a direct two-way exchange of information and a point of contact. Questionnaires reach people unavailable for interviews and allow time to research answers, but are subject to

Table 3.2 Sources of Documents for a Sizing Study

Functional areas	Application systems groups
Computer and network operations	Computer and communications systems groups
Higher management	ADP performance evaluation groups
Software development centers	ADP procurement agencies
Vendors	

Table 3.3 Types of Documents Used in a Sizing Study

Feasibility studies	Policy documents
Program listings	Requirements documents
ADPS management reports	Functional area regulations
Computer operation logs	Technology forecasts (e.g., COMPUTERWORLD article)
Periodic ADPS performance listings	Vendor literature
Periodic workload summary listings	System software manuals
Previous sizing studies	Auerbach Reports (Auerbach Publishers, Inc.)
Previous measurement studies	Datapro Reports (Datapro Research Corp.)
	Letters, memos

misinterpretation, absence of interaction, slow turnaround, and completion at the last possible moment, if at all. If possible use a combination of both techniques.

The following guidelines apply to interviews and questionnaires which are developed to support a sizing study:

(a) The sizing team should structure the interviews and questionnaires in terminology familiar to the people being questioned. Unfamiliar terminology should be avoided or adequately defined.

(b) People's degree of confidence in their answers should be obtained and quantified, if possible.

(c) The interview or questionnaire should be designed to minimize the effort required by the people being questioned. In most cases, their participation in a study is not part of their normal duty.

(d) The people being questioned should be informed of how their answers will be used. This will usually provide a more open exchange of information.

c. Data Reduction and Analysis Tools and Techniques. The data collected for a sizing study must often be processed before it can be used. Data reduction tasks are necessary to report workload characteristics and to provide input for the selected sizing tool(s).

In selecting the data reduction tools and techniques, the following guidance should be used:

(1) When possible, existing data reduction packages should be used in lieu of team-developed ones. The team must trade off the absence of some desired reduction features in an existing package against the time, cost, and possibility of error inherent in package modification or a new development.

(2) The selected tools or techniques should be understood by the sizing team. Many useful and highly effective data reduction packages are available, but "blind application" of these packages can cause large errors in the sizing study. Inputs, procedures, and outputs of a package must be understood by the sizing team, and the accuracy of its results must be determined. Documentation of a data reduction technique or package sometimes provides the information for this understanding.

Statistical techniques can be used to analyze the workload data. Analysis of workload variances, ratios, representativeness and trends is necessary to assure data accuracy and consistency. Many data analysis techniques can be found in statistics textbooks, and these texts should be referenced for selecting the appropriate techniques. Guidance for selecting data analysis techniques includes:

(1) For measured data, analysis of data should include an analysis of the variance of data elements among periods of measurement, a determination of the ratios of data elements and ratio variances among periods of measurement, and a checking procedure for determining data consistency and representativeness of the sample.

(2) For predicted data, analysis of data should include a comparison of projected trends with historical trends, an analysis of the variance of data elements among projections, and a checking procedure for determining data consistency.

(3) The techniques used should be selected with two purposes: (a) assuring that data to be used in the sizing is valid and (b) providing a quantitative assessment of the data accuracy.

d. <u>Sizing Tools and Techniques</u>

(1) <u>Models</u>. Sizing models are used to predict the ADPS required to support a defined workload. Three types of sizing models which can be used are static, analytic and simulation.

(a) <u>Static</u>. Static models usually include a handbook or guide that describes a series of calculations for determining hardware, software, and communication components for a defined workload. This guide may be available prior to a sizing study or may be developed by the sizing team for their particular study. Formulas and factors used in these models are usually based on measured or documented ADPS characteristics, historical data on system performance, and several assumptions about the workload to be sized.

(b) <u>Analytic</u>. Analytic models differ from static models in that more rigorous mathematical techniques are used to develop, verify, and validate the analytic formulae. Also, methods for representing the dynamics of an ADPS' processing of a defined workload are available. This includes representation and prediction of multiprogramming, multiprocessing, processor-I/O overlap, priority structures, and resource queuing effects. Analytic models rely heavily on queuing theory and approximation techniques to represent system dynamics. Because of the numerous and complex calculations involved, these models are typically implemented as computer programs.

(c) <u>Simulation</u>. Two major types of simulation tools are available for sizing: package simulators and discrete-event simulation languages. A package simulator is a set of vendor-supplied computer programs that models one or more computer systems with analytic algorithms, discrete techniques, a library of ADPS characteristics (or factors), and automatic (generated from measurement data) and/or user-defined system activity in a workload definition language. A discrete-event simulation language is a high-level programming language that allows the user to control the structure of the sizing model to whatever extent necessary through the use of specialized simulation programming statements and control mechanisms. Pre-programmed models of hardware and software functions are available. Both types have been used extensively, and simulation models that represent many different ADPS have been developed.

(2) Benchmarks. A benchmark is a compromise to total production operation and is an experimental process of selecting and executing a workload model which approximates the real environment. Benchmarking may be accomplished using the current operational system or a proposed system. Because benchmarking can be expensive, the expected cost and values of the results should be compared before implementation begins. Based on the sizing objectives, one or more testable hypotheses are developed. The benchmark may use live users, actual software, a subset of those users of software, or a synthetic representation of software and/or workload.

(a) Actual Users and Software. "The most accurate and probably most costly method of analyzing and predicting (an ADPS size) is to run the actual system under normal production load and assess its performance. For future planning purposes, current loads can be increased to projected levels and the system performance evaluated."[4] Obviously, the method is limited to studies where the software, workload and ADPS are available for conducting benchmarking experiments. This situation is rare in the early stages of new requirements.

(b) Synthetic Representation. A frequently used method to augment benchmarking is the development of synthetic workload representations. In this method the ADPS alternative that is being sized must be available, but all or portions of the application software and its workload are synthetically represented. This involves creating model jobs, data, and system loads that represent portions of the current and/or the future workload, a process that requires analysis of the software and workload to be represented. Some synthetic benchmark program generators are available from vendors and government organizations. Remote-terminal emulators, that can generate various on-line network loads for testing on-line systems, are also available from vendors for most ADPS's.

(3) Relative Guidance for Sizing Tool Selection. A relative guide is provided below for tool selection based on six factors considered in the macro-analysis. Static models apply to each factor's first attribute, whereas each factor's second attribute applies to benchmarks. Other tools fall between each factor's attribute range. Data Availability: Lacking in detail, estimated as a total; Or, detailed information available on system; Assumptions/Constraints: Numerous, broad, high-level; Or, few, detailed, specific; Timing/Funding: Short study, limited funding; Or, lengthy, costly; Tool Availability: Quickly developed or obtained; Or, may require extensive software development: Accuracy Requirements. Gross approximation of PPB elements ($\pm$25%); Or, detailed estimate of each alternative ($\pm$5%); Desired Results: Estimates of

[4] Air Force Regulation 300-12, Volume I, Attachment 14, p. 43.

alternate costs; Or, estimate of performance for each alternative. In addition to these six factors the sizing tool may be selected based on the tool experience of the team, especially if no one tool or technique dominates the listed factors.

3.3.6 Determine Feasibility of Sizing Study

Prior to planning the details of the study, the sizing team must insure that the study can be completed within the fund and time limitations. This determination is based on the results of the previous analysis (sizing objective requirements, list of alternatives, data availability, assumptions and constraints, and selected tools/techniques) and the use of a skeleton model of the sizing approach. The results of this feasibility analysis must be reviewed by management before the sizing team proceeds with the study. Any sizing team concerns about time, fund, or data constraints should be voiced at this time.

3.4 Plan Detailed Sizing Approach

The information gathered to this point is used to prepare a schedule of sizing tasks and their sequence. At least the following elements must be included in the schedule:

1. Documentation from the previous tasks stating the sizing objectives, data requirements and availability, assumptions and constraints, time limitations, funding constraints, sizing alternatives, desired results, and selected tools and techniques.

2. A description of the sizing experiments needed to size each alternative and achieve the desired results. The experimental design must detail the data sources, analysis procedures, type of desired results (data), and expected accuracy. This is an extremely important element that must be well developed in the sizing approach.

3. A description of each task and subtask to be performed. The tasks and subtasks will fall into one of five categories: (a) obtaining or developing sizing tools and techniques; (b) obtaining data for workload definition; (c) describing sizing alternatives; (d) performing sizing experiments and analysis; and (e) interpreting, validating, and reporting results. (These five categories comprise the remaining steps of the methodology - see Sections 3.5, 3.6, 3.7, 3.8, and 3.9.) Each task or subtask description should include: (a) required input for task and source of input; (b) required output from task and destination (follow-on task) of output; (c) procedure(s), tool(s), and technique(s) to be used; (d) time schedule; and (e) personnel assignments and responsibilities.

4. A set of criteria for evaluating the sized alternatives. This set of criteria should be based on the sizing objective. To the extent possible, the criteria should be quantifiable (cost, performance characteristics, variances, etc.). It may be necessary to weight (quantify or qualify) the criteria based upon their degree of importance. Table 3.4 lists some criteria that might be used in evaluating sized alternatives. At the completion of this step, management should review the sizing approach for consistency with the sizing objective.

3.5 Obtain and/or Develop Sizing Tools and Techniques

The sizing team may need to obtain, develop, modify, enhance or define procedures for use of the selected sizing tools and techniques before performing the sizing experiments. This step can be very time consuming and should be accomplished in parallel with the workload data collection (Section 3.6) and description of alternatives (Section 3.7) tasks. Guidance for this phase can usually be obtained from documentation, vendors, developers, or users of the selected tools and techniques.

The sizing team may also have to verify and calibrate the selected tools and techniques. Verification ensures that a developed method behaves as the sizing team intends, while calibration tests and rectifies the accuracy of the method results with empirical data. A typical calibration includes generating sample data (similar to that being collected for the sizing experiments), conducting a sizing experiment with that data, observing the method and the accuracy of obtained results, and modifying or changing the method if it will not meet the sizing objective. Calibration requires the comparison of an existing workload's performance on an existing ADPS with the method's ability to predict the size and performance of that ADPS, given a description (input data) of the existing workload's characteristics. The extent of calibration that can be performed, of course, is dependent on the availability of an existing workload and ADPS.

Table 3.4 Examples of Criteria for Evaluation

(1) Sized system can process all functional systems and satisfy required turnaround or response times.

(2) Sized system's capacity is expandable for projected future growth.

(3) Sized system can handle unique processing requirements (stand-alone, classified, contingency, specific scheduling constraints).

(4) Sized system can satisfy reliability/redundancy constraints.

(5) Sized system is within cost boundaries (comparative: comparison of alternatives; or absolute: funding constraint).

3.6 Define Workload To Be Sized

3.6.1 Workload Data Elements

The data elements used to describe the workload should be grouped according to a scheme (e.g., by functional area and processing category). Functional area groupings separate the workload by ADP user applications (personnel, civil engineering, operations, maintenance, word processing, etc.); these are the groupings used to predict future workload growth. Processing category groupings identify the workload's mode(s) of processing (e.g., on-line, batch, remote job entry, time-sharing, message-switching); each of these groupings requires a different set of data elements to describe its workload. Usually, both groupings will be used in describing the workload. For example, within a functional grouping may be subgroupings of on-line inquiry and batch.

Within each grouping, two types of data elements are needed: load descriptors and workload characteristics. The number of data elements required for each of these is dependent upon the study's level of detail, desired results, and selected sizing tools and techniques. Table 3.5 provides examples of data elements (and their sources) that might be used to describe three processing categories' loads and characteristics. In selecting data elements for describing a workload, the following guidance should be used:

a. The set of characteristics used to describe a grouping should be independent of a specific ADPS. For example, the amount and characteristics of data to be stored should be specified, and not the number and/or capacity of disk drives. There may be studies, however, where it is more expedient to use certain device-dependent characteristics. This can be done when validated conversion factors are available for translating characteristics across all specific sizing alternatives. If device-dependent characteristics are used, the sizing team must ensure that erroneous conclusions are not made based on the workload characteristics (e.g., "CPU processing times on system A are twice as long as system B for all groupings; therefore, system B can process twice the load of system A," would be an erroneous conclusion). Also, conversion factors must account for pertinent differences in alternative ADPS (e.g., system B may have a processing cycle time that is twice as fast as system A, but system B may devote a much larger portion of its processing to system overhead).

b. The selected groupings should facilitate prediction of future workloads and their characteristics. Predictions can be made by (1) identifying additional groupings and the expected date of their implementation or (2) identifying changes in loads of

Table 3.5. Examples of Workload Data Elements

DATA ELEMENT AND SOURCES		PROCESSING CATEGORY		
TYPE OF ELEMENT	SOURCES	ON-LINE	BATCH	TIME-SHARING (TS)
LOAD DESCRIPTOR	Interviews Questionnaires System Logs Observations User Documentation Scripting Facilities	-Transactions Per Unit Time Per Terminal [Transactions May Be Subdivided into Various Functional Categories]	-Jobs Per Unit Time [Jobs May Be Subdivided into Various Functional or Resource Usage Categories]	-Number of Interactions Per Unit Time [Interactions May Be Subdivided into TS User Categories] -Number of Time-Sharing Jobs Per Unit Time
WORKLOAD CHARACTERISTIC	Hardware Monitors Software Monitors Accounting Logs System Documentation	-Disk I/O's Per Transaction -Input Char/Output Char Per Transaction -Processing (Number Instructions, CPU Time, etc.) Per Transaction	-Resources Consumed Per Job -Program Size -Overlay Counts Per Job	-Resources Consumed Per Interaction -TS Job Sizes -TS Job Memory Residence Times -Number of Characters Per Interaction

existing groupings and when those changes will likely occur or (3) producing predictions of ADPS required to support major variations in expected workload (sensitivity analysis) and reporting the ADPS workload boundaries at fixed performance levels. For example, a future workload at some date may consist of two new functional area groupings and a 10% increase in transaction volume for all existing on-line groupings;or,future workloads may be uncertain.

c. An alternative method of grouping existing workload by similarity in characteristics is cluster analysis. It can be effectively used if little is known about the workflow, functional areas or processing categories, or if the category groupings do not define a stable set of characteristics (e.g., no single set of characteristic values or functions adequately describe the grouping). If cluster analysis is used, the sizing team must not only identify the workload groupings, but must also analyze the groupings and explain the composition of each. This analysis will assist in predicting future loads for each grouping, due to future requirements and growth.

d. Composite measures (e.g., of computer resource usage) should be used with care in describing workload characteristics. These can create problems in describing new requirement workloads, determining alternative system capacities, relating the composite measures across alternative systems, and identifying configuration requirements of each alternative that will process the defined workload. Section 4.5 provides an example of the problems that occur when composite measures are used for sizing.

3.6.2 Application Software Characteristics

The sizing team must do more than just identify the workload for all application programs. Several software characteristics affect when, how, and under what conditions portions of the workload are processed. The major characteristics are as follows:

a. Scheduling. Certain programs can only be processed at certain times or only when certain conditions have been met. These constraints must be represented in the sizing.

b. Dependencies. Often a program cannot be run until one or more other programs have executed. If this occurs frequently and/or the study requires a detailed analysis of sizing alternatives' performance, these dependencies will have to be accounted for in the sizing.

c. Restrictions. Certain programs may require restricted processing because of security, priority, or reliability considerations. These restrictions may require stand-alone processing or other constraints which must be considered in sizing.

d. Maintenance. Many programs use data files that are maintained on regularly scheduled intervals. Maintenance often includes changes in storage media and the organization of the file. These and other actions affect the start time and use of system resources of programs that access these data files.

3.6.3 Communications Load

Communication loads can be described at various levels of detail and should be consistent with the ADP workload definitions. Communication loads might be stated in terms of transactions, messages, packets, characters, or bits of information transmitted within a network. Important characteristics of a communications load include sources, destinations, terminal type, rates, and transmission constraints.

3.6.4 Data Collection

The data elements that were selected for representing the workload and its characteristics are now collected for each defined grouping using the previously selected data collection tools (see Section 3.3.5.b). This collection may include measurement, research, and estimation of the required data. If measurement of existing loads and characteristics is necessary, the following sampling should be applied:

a. The length of each measurement period should be determined by an analysis of the category of workload, the cycles of the workload (i.e., how often does all or a portion of the workload repeat itself), and the sizing study's level of detail. Table 3.6 provides general guidance based on common workload categories and cycles.

Table 3.6 Length of Measurement Period (1 Cycle)

WORKLOAD CATEGORY	MINIMUM	TYPICAL	MAXIMUM
Total*	1 day	1 month	Available history
Batch-Cyclic	Dependent upon cycle occurrence and length		
Batch-Random	8 hours (1 shift)	1 day - 1 week	1 month
On-Line	3-4 hours	1 day - 1 week	1 month
Time-Sharing	3-4 hours	1 day - 1 week	1 month

*No distinction among processing categories is made for sizing.

b. The number and kind of measurement periods depends on the stability of the workload, the defined level of detail, accuracy requirements, alternate sources, available time and funds, and the number of workload groups. The sizing team should understand the workload to be measured, so that a minimum number of periods are needed to collect a workload description. It is the team's responsibility to select an adequate number of periods in order to meet the sizing objective, and then to carefully conduct the required data collection. For example, if a maximum on-line response time is specified, the measurement of on-line workload must be at a peak period. If, however, no critical performance constraints exist, measurement during or near an average level of activity should suffice.

Guidance for future workload data collection includes:

a. Future workload should be estimated in the same terms (load descriptors, characteristics) as the current workload (if one exists). The total workload requirements can then be sized using a single set of sizing tools and techniques.

b. Groupings for the current workload should be carefully examined for similarities with future workload estimates. When possible, future workload should be estimated in terms of current workload groupings. For example, a new online system's workload might be estimated as having the same characteristics of an existing on-line system, but with 80% of the existing system's transaction volume.

The information necessary to describe the application software characteristics is best obtained from functional analysts, programmers and program documentation. It is the sizing team's responsibility to obtain this information through interviews, questionnaires, or existing documentation.

Some communications load characteristics can be extracted directly from the workload definition and user requirements. Transaction volumes, sizes, and responses should be identified; additionally, sources of remote activity should be stated for the communication sizing. Unique terminal requirements (hard copy, optical-character read, hand-held data entry, etc.) must also be collected. Besides extracting this data from the workload definition, the sizing team can also gather information from functional users (through interviews and questionnaires).

3.6.5 Workload Representation

Often it is necessary for the sizing team to reduce, structure, combine, summarize, extrapolate or reformat portions of the collected data. This is necessary to provide a manageable, but representative, description of the workload to be sized. This process, called "window analysis" scales the workload through a "window", i.e., reduces the workload to one that is feasible and practical to model or benchmark and that nonetheless represents the larger original workload.

In representing the workload, the following guidance should be used:

a. The workload representation must be consistent with the study objective and constraints. If a portion of the total workload has a time or resource constraint, then it must be represented. If the study is to determine a required capability for all workload, then peak processing periods should not be solely represented (nor should low processing periods).

b. The sizing team should <u>document</u> their total description of the workload. The procedures, assumptions, and results of this process should be readily available for review. A report on the feasibility of processing new requirements must document the workload to be processed. Workload documentation quantitatively expresses (1) new requirements, (2) summaries of functional and processing category requirements, (3) the level of detail at which workload data was obtained, and (4) the accuracy and possible variances in the collected data. The workload representation is, of course, key to the success of the sizing effort.

3.7 Describe Sizing Alternatives

A detailed description of the sizing alternatives must be developed for the sizing experiments. This description may be a set of hardware, system software, and communication component characteristics, or an actual benchmark configuration. The exact method of describing each alternative depends upon the sizing tool(s) being used in the study.

3.7.1 Hardware

a. <u>Data Elements</u>. Hardware characteristics for each alternative should be stated in terms of device speeds, capacities, and connectivity. It is <u>not</u> recommended that the sizing team decribe hardware characteristics using performance measures (composite measures, jobs processed per hour, transactions processed per second, etc.). While it simplifies the sizing technique to describe the hardware characteristics as workload rates (e.g., jobs processed per hour), it is almost impossible to show that the workload units equate to the hardware rates (e.g., the workload's jobs do not necessarily equate to hardware's job processing rate).

b. <u>Collection</u>. Vendor literature and various other documents on ADPE should provide the sizing team with the required hardware characteristics. Important elements which the team must collect for each hardware alternative include device types, device speeds, device capacities, configuration ranges (min, max), device connectivity, and limitations.

c. <u>Representation</u>. The hardware, software, and communication component characteristics are used to represent ADPS alternatives. The characteristics of one component affecting the characteristics of another (e.g., system software overhead can reduce the speed and capacities of hardware) must be included in each ADPS alternative description. Two approaches can be taken to describe each alternative's characteristics. The first approach is that the sizing team can use a system with which they are familiar. A better definition of the component characteristics can thus be provided. An advantage is that sizing can then be costed on the basis of known vendor price schedules. Assuming that vendors are competitive, this sizing should represent the approximate dollars that will be expended, and general performance of the sized ADPS. The results of this sizing approach should be used primarily for programming, planning, and budgeting. One disadvantage of this approach is that the sizing can easily bias procurement actions in favor of the vendor used in the sizing. A second disadvantage is that vendors may, in general, be competitive, but the unique aspects of a workload may be well suited to a particular ADPS other than the one used for the sizing.

The second approach is to describe characteristics of representative components that satisfy sizing alternatives. This approach makes it possible to "fine tune" the characteristics of the ADPS needed to support the defined workload. The disadvantages are in describing the characteristics and in accurately costing the sized ADPS. Although the characteristics of the sized ADPS may be well defined with this approach, it is often difficult to relate these pseudo-characteristics to available systems.

It is generally best to use the first approach when PPB considerations or system capacity considerations are of prime interest and to use the second when portraying a size without any bias toward a specific vendor is most important.

3.7.2 System Software

a. <u>Data Elements</u>. ADPS alternatives must include consideration of system software and its impact on the total

ADPS performance. At a minimum, the overhead (resource usage) imposed by the system software must be included in each sizing. As more detail is added to the study and more information on each alternative's performance is required, the system software's features and logic must be represented in the sizing tools and techniques. Major features which might be analyzed include: (1) I/O handling; (2) user resource request handling; (3) job scheduling, initiation, and termination; (4) job execution management (task switching, queuing, swapping, etc.); (5) data management; (6) network management; and (7) special features (sort packages, resource accounting, etc.).

b. Collection. The information that is needed to describe alternative system software must come from sizing constraints, assumptions, vendor literature, texts on system software, instrumented benchmarks, software monitors, and application systems' requirements. The sizing team should list the minimum features which will satisfy the sizing constraints, assumptions, and application system requirements. Vendor literature and texts on system software can then be used to select the features and ensure that the required set of features is available.

c. Representation. A representative set of system software (representing no particular vendor's software) or a specific vendor's system software can be used in describing ADPS alternatives. The description of system software alternatives obviously depends on the selected sizing technique (model or benchmark). Section 3.7.1.c provides further guidance on which representation to use.

3.7.3 Communication Components

a. Data Elements. The hardware and software needed to handle data transmission between a remote user and central computer site or multiple central computer sites comprise the communications network. Communication devices include lines, terminals, modems, concentrators, bridges, communication front-ends, and multiplexors. Communication software includes packages for network control, accounting, error detection and correction, code conversion, recovery, and security.

b. Collection. The communication device characteristics can best be collected from vendor literature, and selected texts. The characteristics should be stated as basic capabilities: speed, capacity, connectivity, reliability, queuing discipline, etc. As stated in Section 3.7.1.c, either specific vendor device characteristics or representative characteristics can be identified for each alternative. Communication software procedures are similar to those of system software (Section 3.7.2.b). Human factors are obtained by interviews and observation, operator documentation, or prototyping.

c. Representation. The communication devices must be described such that their characteristics can be input into the sizing tools and a communications network identified. If extensive communication requirements must be identified and sized, a communications engineer should be a sizing team member. Line speeds, terminal characteristics, and other device characteristics can be obtained from literature surveys, but representing the networking of these devices requires additional expertise. Communication software representation, like system software representation, depends heavily on the sizing technique (Section 3.7.2.c).

3.8 Perform Sizing and Analysis

The sizing experiments consist of applying the developed sizing tools and techniques (Section 3.5), using the defined workload, software, hardware, and communications data (Sections 3.6 and 3.7). The number of sizing experiments that are performed depends upon the time available, resources, desired results and required degree of accuracy. Before and during these sizing experiments, the following guidance should be applied:

1. Prior to conducting the first sizing experiment, the sizing team should review the data from all previous steps. This review should last anywhere from two days to several weeks, depending upon the study's level of detail and should address the following activities:

a. Review the sizing objective and ensure that experiment results will satisfy that objective.

b. Review each alternative and ensure that it is still viable.

c. Review all identified assumptions and constraints; identify where they are considered in each sizing experiment; ensure that they are being considered correctly.

d. Review the calibration and verification tests that were performed for the sizing tools and techniques. Ensure that the tools behave as desired and identify the confidence that can be placed in the tools' results. (Review calibration results.)

e. Review all data collected to define the workload, software, hardware, and communications. Ensure that the required input data for the sizing are available. Ensure that data element definitions are consistent (e.g., make sure that transactions defined in the ADPS workload are equivalent to transactions defined in the communications workload).

f. Make a final review of the sizing procedures that will be used during each experiment. Clarify any misunderstandings among team members.

2. The evaluation criteria that were developed during the detailing of the sizing approach (Section 3.4) must be used in selecting the best ADPS. Each sized ADPS should be evaluated against each criterion. If possible, quantitative measures should be used in this evaluation. Some measures such as costs, response times, and resource utilization levels are straightforward, while others such as level of security, satisfaction of user requirements, and ease of software conversion may require the sizing team's subjective weighting of each alternative. The sizing team must use the results of this evaluation to first identify which alternatives satisfy all criteria and then, from that subset, to select the alternative which best satisfies all criteria. The approach and results of this evaluation should be well documented.

3.9 Interpret, Validate and Report Results

After the sizing experiments are completed, the sizing team must carefully review, interpret, and validate their results. The team must consider the sizing objective(s), assumptions, and constraints when reviewing the results. Sensitivity of input data (workload, descriptions of alternatives), assumptions, and constraints should be carefully analyzed. Because of the uncertainties that are inherent in sizing studies, the team must interpret the results with care. Once the results have been reviewed and interpreted, the sizing team must validate that their interpretation is consistent and accurate. Any unknowns or uncertainties must be recorded for the users of the sizing results.

Documentation of the study results and the sizing techniques must receive considerable attention. It is the final documentation which will be used in future decisions and actions regarding the selected ADPS and its estimated size. The following guidance should be used in preparing this documentation:

1. Report the results in terms of the sizing objective. Do not increase the document size by reporting additional results that do not support or relate to the sizing objective. Supplementary information to support the sizing results should also be provided. Any other information from the sizing study should be retained in case further results or details on the sizing are required.

2. Report the sizing team's confidence in the results, and identify how uncertainties in the study might affect the results. Five areas of uncertainty which should be addressed are (1) assumptions and constraints, (2) collected data for study, (3) tool/technique accuracy, (4) workload projections, and (5) subjective interpretation.

3. Structure the sizing report to briefly cover all steps in the sizing process. Highlight those sections which specifically satisfy the sizing objective. Provide results in figures, graphs, and charts, when possible. Excessive words only detract from the sizing results. Table 3.7 presents an outline which can be used in structuring the sizing report.

4. Sizing Problems/Examples

4.1 Learning From Past Studies

Sizing studies are certainly not new. Government and Industry have conducted many sizing studies to estimate the feasibility, cost, and performance of proposed new or enhanced ADPS. However, it is often difficult (if not impossible) to obtain the details of studies that are similar to a new sizing requirement. This can lead to duplication of effort and the recurrence of sizing mistakes. To reduce this duplication, this section discusses selected sizing studies and some of the major problems that have arisen. Section 5 provides references for other sizing studies that have been conducted.

Table 3.7. Example Outline for Sizing Report

i ABSTRACT

Summary of sizing objective, sizing results, and study recommendations.

ii TABLE OF CONTENTS

iii LIST OF FIGURES AND TABLES

I. <u>INTRODUCTION (MACRO-ANALYSIS)</u>

- A. SIZING STUDY OBJECTIVE
- B. ASSUMPTIONS/CONSTRAINTS
- C. STUDY DATA AND SOURCES
- D. LIST OF ALTERNATIVES

II. <u>SIZING APPROACH</u>

- A. TEAM AND TASK DESCRIPTIONS
- B. DESIGN OF SIZING EXPERIMENTS
- C. TOOL/TECHNIQUE DESCRIPTIONS
  1. Selection
  2. Verification/Validation
- D. WORKLOAD DEFINITION
  1. Sources
  2. Approach
- E. SOFTWARE, HARDWARE, COMMUNICATION DEFINITION
  1. Sources
  2. Approach
- F. CONDUCT OF SIZING EXPERIMENTS

III. <u>SIZING RECOMMENDATION</u>

- A. WORKLOAD SUMMARY
- B. RECOMMENDED ADPS DESCRIPTION
  1. Software
  2. Hardware
  3. Communications
- C. SPECIAL CONSIDERATIONS
  1. Impact of Changing Assumptions/ Constraints
  2. Impact of Workload Variance
  3. Confidence in Sizing Tools/ Techniques

IV. <u>ALTERNATIVE ANALYSIS</u>

- A. EVALUATION CRITERIA
- B. EVALUATION APPROACH
- C. ANALYSIS RESULTS

<u>APPENDICES</u>

A. DETAILED WORKLOAD DEFINITION

Tables, graphs, figures, and narrative descriptions should detail the workload and its characteristics. Functional area and processing category groupings should be described.

B. ALTERNATIVE SIZINGS

For each alternative, provide a detailed description (commensurate with sizing level of detail) of the sizing results. Report the analysis of variances for each alternative (impact of changing assumptions/constraints, impact of workload variances, etc.).

4.2 Constraining Sizing Alternatives

Predetermining the viable alternatives is one of the common mistakes made in sizing studies. Although a complete set of alternatives must be identified, such identification should occur within the normal flow of the sizing methodology (see Section 3). Early determination of which alternatives are viable often directs the sizing study toward justifying a particular sizing alternative, rather than toward evaluating all the alternatives that might satisfy the sizing objective, assumptions, and constraints. A recent technical evaluation of a sizing requirement concluded, for example, that:

> "Not all reasonable alternatives have been identified. The approach and methodology used to evaluate the system alternatives are inadequate. Alternatives have been discarded without proper study and analysis. Statements about capabilities and capacity are made without supporting analysis. No feasibility studies or detailed information of the identified alternatives exist."[5]

The sizing team's premature selection of the "best" alternative led to an unsatisfactory study. Had the sizing team carefully treated each alternative equally and analyzed all alternative sizings, the study would have provided the information necessary for the decision makers.

An alternative that appears to be the best for the sizing organization may be unacceptable because of constraints identified during the review cycle. If the team "justifies" only one alternative, instead of evaluating several, the sizing will probably have to be repeated. Just that happened in the study mentioned above. The study attempted to justify a sole-source procurement (a most difficult task for any sizing team) and not all viable alternatives were evaluated equally. The study's conclusion that the sole-source procurement had "the greatest assurance of providing a comprehensive solution to the problems identified in terms of operational timing, technical risk, mission support, compatibility, and cost" was not justified by an analysis of sizing alternatives.

[5]Technical Evaluation of Reference Study #1 (Referenced studies will not be specifically named, since that would add little to this discussion).

4.3 Poor Workload Definition

Two essential elements of a sizing study are (1) adequate workload definition and (2) a statement of workload variance. Adequate workload definition requires that the sizing team be able to <u>identify</u> the workload to be sized, <u>collect</u> it, <u>analyze</u> it, and <u>represent</u> it in a form comparable to the sizing method. Stating the workload variance identifies the confidence the sizing team has in the workload definition and forms the basis for sensitivity analysis.

Both adequate workload definition and a statement of variance have sometimes been neglected in past studies. The former is essential no matter what type of sizing is to be performed. For example, the study evaluation mentioned in Section 4.2 maintained that:

> "No comprehensive workload definition or system sizing studies have been performed. Agency personnel stated that requesting prototype equipment for evaluation negated the requirement that workload definition and sizing studies be conducted. Agency position is that the studies would be developed during the prototype evaluation period as necessary. <u>The workload definition and sizing studies should be the foundation for the entire evaluation</u>. A detailed prototype evaluation plan is mandatory before the acquisition or installation of any system. Without such a plan, no objective evaluation is possible; indeed, it is highly likely that an unorganized test will mislead if it reacts to occurrences as they happen rather than a predefined plan."

This sizing study defined existing workloads at a very high level (e.g., wall clock hours, "average" transaction activities) but did not address such key data as desired transaction response times, output transaction volumes (by type), and differentiation between transaction types. The study's methodologies and data collection techniques were undocumented, and no workload variance estimates were reported or analyzed for alternative sensitivities. All this made the review cycle of the study difficult and slow. Several iterations of reworking the study were necessary to improve its sizing approach.

4.4 Inappropriate Level of Detail

Sizing studies are frequently either much too detailed or not nearly detailed enough, and either extreme slows the implementation of new requirements. An insufficiently detailed sizing study normally requires that additional information be provided before decisions on future ADPS can be made. Unnecessary detail will slow the sizing effort and review of sizing results.

An example of lack of detail in a sizing to support a requirement was recently documented. The study requested upgrading CPUs at two sites and attempted to justify the upgrades with a workload analysis that showed increasing CPU processing times at both sites. The study did not, however, analyze the alternatives (no change, upgrade peripherals, upgrade CPU and associated peripherals) and their abilities to process the workload. As a result, the reviewing authority responded by stating that:

> "Our review ... has disclosed a lack of detail surrounding the definition of the specific problem at Site 1 and Site 2 and, given that, the specific alternative(s) available for the solution of the problem. The study states that systems at both sites are CPU bound but fails to adequately justify this assertion."[6]

It was recommended that workload analysis and sizing studies should be performed at both sites. As a result, approval of the study was delayed.

Excessive detail often results from using sizing methods appropriate for well-defined, existing workloads to size ill-defined future workloads. An attempt to increase the sizing accuracy simply by using precise techniques can introduce an undesirable level of detail and obscure potential workload variances. It is better to clearly and quickly report large variances in the sizing results than to dwell on sophisticated approaches that lend nothing to the desired sizing objective.

An appropriate level of detail is extremely important for preparing a Live Test Demonstration (LTD). Too little detail in the LTD will lead to incomplete testing of bids. Excessive detail will cause misunderstandings among the vendors and make the conduct of fair tests very difficult. Further guidance in preparing an LTD at the appropriate level is provided in FIPS PUB 42:1 (see Section 5).

[6]Response to Reference Study #2.

4.5 Improper Tool Selection

More than one set of sizing tools can usually be applied in any sizing study. Using the guidance provided in Section 3.3.5 (Select Tools/Techniques) may lead to the selection of different sets of tools for similar studies. However, the sizing team must always guard against selecting a tool not suitable for the sizing study.

One example of a potentially inappropriate sizing technique is analyzing composite measures (e.g., computer resource units [CRU's], standard units of processing [SUP's] etc.). Although composite measures describe various jobs' uses of computer resources and provide a vehicle for costing a job equitably, such measures are usually not ideal candidates for sizing studies. Potential problem areas with using composite measures in a sizing analysis include:

(1) Describing new workload requirements

(2) Determining alternative system capacities

(3) Statistically relating composite measures across different alternative systems

(4) Identifying the required configuration for each alternative to process the workload

(5) Instability of a composite measure over the time of the study.

All these problem areas are illustrated in the following example.

A recent sizing study[7] was based on an analysis of composite units. An estimate of outyear workloads (in terms of composite results) was compared to calculated composite unit capacities for the study alternatives, but the study documented no assumptions, constraints, or estimates of error. This approach had several deficiencies.

[7]Reference Study #3.

(1) The workload was defined by determining the average job composite Unit/Hr rate for the existing workload. That rate was then multiplied by user estimates of occupancy hours for each outyear. This assumed (a) that the average composite rate would not change over time, (b) that the composite rate was a good measure of workload, (c) that the user could accurately estimate occupancy hours, and (d) that the composite rate was configuration independent. The first assumption, which was equivalent to assuming that average job resource requirements would not change, should not have been made without some analysis of the future workload's characteristics. The second assumption did not consider the effect of different workload characteristics. For example, two outyear projections of workload could have identical composite rates, but be significantly different. The first might consist of CPU-bound jobs with very little I/O activity, and the second might be comprised of I/O-bound jobs with low CPU activity. Although the systems sized to support these two workloads might be quite different, the composite measure of workload would not identify such a difference. The third assumption could lead to large variances that should be recognized and studied through sensitivity analysis. Although requesting estimates of future requirements is necessary, it can lead to large errors. These errors are compounded by estimates based on such an aggregate measure as occupancy hours. The fourth assumption gives undue credibility to composite measures as absolute gauges of workload, ignoring hardware and software constraints.

(2) The calculation of composite capacities for the study alternatives assumed a balanced use of all system resources. Maximum composite factors were calculated for the CPU, core, tape, disk, and mass storage. An efficiency factor was then multiplied by the total of the composite factors to identify each alternative's maximum capacity in composite limits. An analysis of existing systems showed that, because a sustained balance of resource usage was not possible, it could not be expected that the sum of these maximum composite rates would ever be achieved. A percentage factor was applied to the sum of the maximum composite rates to account for this fact. That factor, however, was based upon an existing workload and did not account for the effect of future workloads on the system's balance.

(3) In order to evaluate alternatives, the total workload demand in composite units were compared to each alternative system's composite unit capacity. This approach does not necessarily alleviate a saturation condition. For example, a system with a "capacity" of 450 Units/Hr could be increased to a "capacity" of 1500 Units/Hr by adding core and a faster CPU. If the workload required 800 Units/Hr, the new system should be able to process the workload (on the basis of a composite unit analysis). However, the workload could be heavily I/O bound and have its throughput seriously degraded by a lack of I/O channels and devices.

Sizing tools that define the workload and system alternatives in basic ADP resource units and that analyze alternatives at the component level usually provide more meaningful and more accurate sizings. Even when only gross estimates of workload are available, these approaches can lead to a better understanding of requirements and the effect of variances on the estimates. A static or analytic sizing would have been more appropriate than composite unit analysis for the sizing just discussed. Such a study could have analyzed each alternative at the component level and provided variances about the estimates.

4.6 Poor Presentation of Results

Poorly documented results can make an excellent sizing study ineffective. Besides following good report writing procedures, the sizing report should include at least:

(1) A concise summary that presents sizing objectives, assumptions, constraints, data sources, approaches, alternatives, and results

(2) A firm recommendation supported by the study

(3) An appendix that details the study and is formatted like the sizing methodology in Section 3

The inadequately presented results in one study undermined an excellent and thorough workload analysis, a good saturation analysis, the proper use of existing tools, and a detailed evaluation of various alternatives. The study's results were not concisely summarized. The opening section of the study stated that, "The amount of direct time indicates that the system is close to saturation, and the situation is close to being critical.[8] The study added that "saturation may be reached in the very near future." The objective stated that "this request is to relieve the approaching saturation condition before it significantly affects support capabilities to all functional users. Under present circumstances, the desired objectives can only be obtained through ... the installation of another system at the central site. This will definitely relieve the pressure of the present workload and eliminate the eventual saturation as a result of workloads projected for the future."

The wording of this summary left many questions in the minds of the reviewing authority and required a follow-up letter that more explicitly stated future workload, sizing results, and when the new workload impacts would occur. The summary's wording immediately raised questions such as: (1) What is saturation? (2) When will saturation occur? (3) What workloads will be added to cause saturation? (4) Why does the objective statement also provide a solution? (5) What functional user support capabilities will be affected?

The sizing team should always try to minimize such questions by reporting the results clearly, thoroughly, and accurately.

This section has discussed only a few of the many problems which can occur in a sizing study. Many more have been experienced, and sizing teams should seek out these problem areas prior to experiencing them in their own studies. Section 5 provides a list of references that can assist sizing teams in identifying other problem areas. An intensive example, illustrating most of the techniques described, is available by writing to the authors.

5. Sizing References

5.1 General

All tools, techniques, previous applications, guidelines, and approaches to sizing are far too numerous to document here. The sizing team must complement the methodology and guidance presented in this document with reference to other sizing-related documents and assistance from centers that have experience in sizing. This section lists major references for additional sizing information.

Sizing references that can assist the sizing team in preparing and conducting a sizing study include:

[8] Reference Study #4.

1. AFR 300-12, Volume I, Chapters 2,4,5,7,8, and 9 and Attachments 1,2,3, 15,17,24,25, and 27.

2. Bronner, L. Capacity Planning, An Introduction. Washington, D.C.: IBM Systems Center, January 1977.

3. "Computer Performance Planning and Control." Ideas for Management, 1974, pp. 60-66.

4. Cooley, B. "Documenting Simulation Studies for Management Use." Winter Simulation Conference, Volume 2, 1977, pp. 742-746.

5. DeMarco, T. "Breaking the Language Barrier, Part I" COMPUTERWORLD, August 7, 1978, pp. 19-27.

6. Management Guidance for Developing and Installing an ADP Performance Management Program. Washington, D.C.: General Services Administration, July 1977.

7. "Predicting the Effects of Hardware Changes." EDP Performance Review, February 1974.

8. Timmreck, E. M. "Computer Selection Methodology." Computing Surveys, Volume 5, No. 4, December 1973, pp. 199-221.

5.2 Sizing Tools/Techniques

The selection of proper sizing tools and techniques requires a thorough review of the many available references. The following list is a sampling of available sizing tool and technique references:

5.2.1 Data Collection

a. Carlson, G. "A Guide to the Use of Hardware Monitors." EDP Performance Review, September 1976, pp. 1-8; October 1976, pp. 1-7.

b. Deese, D. R. "Experiences with Measurement as an Aid to Simulation." SHARE, December 1973, pp. 361-374.

c. Desiderio, L.; Saloky, D.; and Wasserman, A. Measuring Computer Performance for Improvement and Savings. Coopers and Lybrand, 1974.

d. Drummond, M. E., Jr. Evaluation and Measurement Techniques for Digital Computer Systems. Prentice-Hall, Inc., 1974.

e. "Evaluation and Comparison of Software Monitors." EDP Performance Review, February 1976, pp. 1-9.

f. Gemar, W.. "Improved Performance for Less Cost." Data Processing, Volume 16, July-August 1974, pp. 225-228.

g. Morris, J. A. "Hardware Measurement - Past, Present, and Future." SHARE, December 1973, pp. 308-332.

h. Nutt, G. J. Computer System Monitoring Techniques. NTIS, February 1973.

i. Svobodova, L. Computer Performance Measurement and Evaluation Methods: Analysis and Application, American Elsevier, 1976.

5.2.2 Models

a. Analysis of Some Queuing Models in Real Time Systems. IBM Journal GF20-0007-1, September 1971.

b. Allen, A. O. ""Performance Analysis and Capacity Planning." Presentation Notes at SHARE 50, Session C113, March 1978.

c. Bell, T. E. "Objectives and Problems in Simulating Computers." Proceedings of FJCC, 1972, pp. 287-297.

d. Clark, C. T. and Schkade, L. L. Statistical Methods for Business Decisions. Southwestern Publishing Co., 1969.

e. Daniel, C. Applications of Statistics to Industrial Experimentation. John Wiley & Sons, Inc., 1976.

f. Emshoff, J. R. and Sisson, R. L. Design and Use of Computer Simulation Models. The MacMillan Co., 1970.

g. Feller, W. An Introduction to Probability Theory and Its Applications. John Wiley & Sons, Inc., 1950.

h. Fishman, G. S. Concepts and Methods in Discrete Event Digital Simulation. John Wiley & Sons, Inc., 1973.

i. Freiberger, W. (ed.). Statistical Computer Performance Evaluation. Academic Press, 1972.

j. Hillier, F. S. and Lieberman, G. K. Operations Research. Holden-Day, Inc., 1974.

k. House, W. C. Operations Research. Auerbach Publishers, Inc., 1972.

l. Jain, A. K. "Statistical Approaches in Computer Performance Evaluation Studies: A Tutorial" Supplementary Proceeding of CPEUG Meeting 14, 1977.

m. Karplus, W. J. "The Spectrum of Mathematical Modeling and Systems Simulation." Mathematics and Computers in Simulation XIX, 1977, pp. 3-10.

n. Kershenbaum, A. and Chou, W. "A Unified Algorithm for Designing Multidrop Teleprocessing Networks," IEEE Transactions on Communications, Volume 22, No. 11, November 1974.

o. Kiviat, P.. J.; Villanueva, R.; and Markowitz, H. M. SIMSCRIPT II.5 Programming Language. CACI, Inc., 1973.

p. Kleinrock, L. Queuing Systems, Vol. 2: Applications. John Wiley & Sons, Inc., 1976.

q. Kosy, D. W. The ECSS II Language for Simulating Computers Systems. The Rand Corporation, R-1895-GSA, December 1975.

r. Martin, F. F. Computer Modeling and Simulation. John Wiley & Sons, Inc., 1974.

s. Pritsker, A. B. The GASP IV Simulation Language. John Wiley & Sons, Inc., 1974.

t. Reiser, M. "Interactive Modeling of Computer Systems." IBM Systems Journal, No. 4, 1976, pp. 309-327.

u. Shannon, R. E. Systems Simulation: The Art and Science, Prentice-Hall Inc. 1975.

v. Schriber, T. J. Simulation Using GPSS. John Wiley & Sones, Inc. 1974.

w. Wagner, H. M. Principles of Operations Research. Prentice-Hall, Inc., 1969.

x. Wyatt, J. B. "Computer Systems: Simulation." The Information Systems Handbook. Chapter 19. Dow-Jones-Irwin, Inc., 1975.

5.2.3 Benchmarks

a. AFR 300-12, Volume I, Attachment 15.

b. Benwell, N. (ed.). Benchmarking, Computer Evaluation and Measurement. Washington, D.C.: Hemisphere Publishing Corp., 1975.

c. "DPSC Interim Benchmark Project Report." Naval Facilities Engineering Command, Facilities Systems Office, April 1975.

d. Guidelines for Benchmarking ADP Systems in the Procurement Environment, FIPS PUB 42-1. Washington, D.C.: National Bureau of Standards, May 1977.

e. General Services Administration. Summary of the NBS/GSA Government Workshop on Remote Terminal Emulation, Report CS76-1, Washington, D.C.: GSA/DJS, June 1976.

f. Handbook for Preparation of Vendor Benchmark Instructions. Department of the Navy, ADPS Selection Office, October 1976.

g. Joslin, E. Computer Selection, Addison-Wesley, 1968.

h. Watkins, S. W. and Abrams, M. D. Survey of Remote Terminal Emulators. NBS Special Publications 500-4, April 1977.

i. Walkowicz, J. L. Benchmarking and Workload Definition: A Selection Bibliography with Abstracts. Washington, D.C.: Government Printing Office, November 1974.

j. Walters, R. E. "Benchmark Techniques: A Constructive Approach." Computer Journal, February 1976, pp. 50-55.

k. Wyrick, T. F. and Findley, G. W. Incorporating Remote Terminal Emulation into the Federal ADP Procurement Process, Proceedings of CPEUG, this issue.

5.3 Workload Definition

References that can assist the sizing team in conducting the workload definition step (discussed in Section 3.F) include:

a. Agrawala, A. D. and Mohr, J. M. "A Model for Workload Characteristics." Symposium on the Simulation of Computer Systems III, August 12-14, 1975 pp. 9-18.

b. Crothers, C. G. "Workload Determination and Representation for On-Line Computer Systems." MITRE Tech Report TR-2682, June 1974.

c. Ehrenberg, A. S. C. Data Reduction. John Wiley & Sons, Inc., 1975.

d. Ferrari, D. "Workload Characterization and Selection in Computer Performance Management." Computer 5:4, July-August 1972, pp. 18-24.

e. Gudes, E. and Sechler, C. "Measures for Workload and the Relation to Performance Evaluation." Proceedings of Computer Performance Evaluation Users Group, September 23-26, 1975, pp. 115-221.

f. Johnson, R. R. "On the Generation of a Demand and Batch Workload Model." MITRE Tech Report MTR-4561, August 1973.

g. Sreenivasan, U. and Kleinman, A. J. "On the Construction of Representative Synthetic Workloads." MITRE Tech Report MTR-143, March 1973.

5.4 Describe Sizing Alternatives

References that can assist the sizing team in conducting the software, hardware, and communication definition step (discussed in Section 3.G) include:

a. Auerbach Computer Technology Reports, Auerbach Publishers, Inc., 1977.

b. Datapro 70, Datapro Research Corporation, Delren, N.J., 1977.

c. Datapro Reports on Minicomputers, Datapro Research Corporation, Delren, N.J., 1977.

d. Datapro Reports on Data Communications, Datapro Research Corporation, Delren, N.J., 1977.

e. Datapro Directory Software, Datapro Research Corporation, Delren, N.J., 1977.

f. Doll, D. R. Facilities, Networks and Systems Design, John Wiley & Sons, 1978.

g. EDP Performance Review, Applied Computer Research Publishers, Volumes 1-6, 1973-1978.

h. Ferrari, D. (ed.). Proceedings of the Performance of Computer Installations: Evaluation and Management, North-Holland Publishing Co., 1978.

i. Ferrari, D. (ed.). Computer Systems Performance Evaluation, Prentice-Hall, Inc., 1978.

j. General Services Administration. User's Guide, Supplement #2 to Teleprocessing Services Program special notice. 1978.

k. Green, P. and Lucky, R. (eds.). Computer Communications, IEEE Press, 1974.

l. Martin, J. Systems Analysis for Data Transmission, Prentice-Hall, Inc., 1972.

m. Proceedings of the Computer Performance Evaluation Users Group, National Bureau of Standards Publication, Meetings 1-15, 1971-1978.

n. Proceedings of the Computer Measurement Group (formerly BBUG), Volumes 1-9, 1968-1978

o. Proceedings of the European Computing Conference on Computer Performance Evaluation, London, 1972, 1974, 1976, On-Line Conferences Ltd., Uxbridge, U.K.

p. Schwartz, M. Computer-Communication Network Design and Analysis, Prentice-Hall, Inc., 1977.

q. SHARE Computer Measurement and Evaluation Newsletter No. 1-44, 1969-1978.

r. SIGMETRICS (ACM) Performance Evaluation Review Newsletter, Vol 1-7 1972-1978.

s. Stimler, S. Data Processing Systems: Their Performance, Evaluation, Measurement and Improvement, Motivational Learning Programs, Inc., 1974.

HUMAN PERFORMANCE EVALUATION IN THE USE OF FEDERAL COMPUTER SYSTEMS: RECOMMENDATIONS

Mark A. Underwood*

Navy Personnel Research and Development Center
Code P204 - Bldg. 330
San Diego, CA 92152

There has been increased awareness in recent years of the high cost of non-hardware items in the Federal ADP budget in contrast with decreasing costs for much of the hardware. More attention is being given to software development costs, systems design practices, automatic program testing, and the like. Particular commercial and military systems effectiveness and life cycle costs now take into consideration such factors as part of the planning process. It is suggested that not enough attention has been given to measurement of human performance variables as part of the systems procurement and systems evaluation phases of Federal ADP programs. Recommendations are made for the incorporation of such measures along with conventional hardware/software performance measurement.

Key words: Computer performance; federal systems evaluations; human performance measurements; psychology of computer systems usage.

1. Introduction

Proliferation of low-cost computer systems in the Federal Government may well reduce or hold the line on overall computer hardware costs. This turn of events will draw attention to a steady increase in personnel costs associated with the operation, maintenance, programming, and use of computer systems. Accordingly, there has been some research in the field of the cognitive and human factors involved in programming. Some of the outgrowths of this research have been "software science" (Halstead), "software physics" (Kolence) and the field of structured design. The popularization of structured programming was spurred, seemingly, by a realization that the sophistication of machines being manufactured exceeded human capabilities to utilize them efficiently.

The Federal Government, some have maintained, has been a stimulus for growth in the area of computer performance evaluation (CPE) because of the role that CPE can play in improving utilization of existing computer systems, and the role it can play in Federal acquisition actions. Most of the effort which has been publicly documented has dealt with performance evaluation of hardware configurations, and with software as a system interacting with a collection of hardware resources. Human resources played an indirect role in such measurements: they produced the software, or they were consumers of the hardware-software configuration (generated n transactions per minute, etc.). As a recent review article indicated [1][1], even the study of

*The opinions and assertions contained herein are those of the writer and are not to be construed as official or reflecting the views of the Navy Department.

[1]Figures in brackets illustrate the literature references at the end of this paper.

software science is limited to measures which can be computed automatically from a computer program (e.g., during compilation), which by its very nature limits the human factors which can be measured. This emphasis upon "programmer behavior" as opposed to "user behavior" is a prevalent distinction--software development rather than software usage costs receive most of the attention. However, as the user community broadens in scope and size, and as dependence upon systems software, documentation and maintenance aids grows, it can be expected that a more elastic notion of measuring the effective utility of a computer system (or a network of them) must be adopted. To carry this argument to its extreme, one can imagine a computer system whose hardware and software have been well-designed, well-implemented, are highly complementary in terms of their net measurable hardware utilization, yet which requires enormous cost to use effectively because of deficiencies in the user-machine interface. In short, a more comprehensive notion of a computer system as a utility must be used: to include space, power [2], maintainability, training for users, programmers and maintenance technicians, organizational impact, and cost for documentation of programs, systems software, and electronic and mechanical items.

The emphasis of this discussion is to broaden the notion of computer system performance measurement. This can be accomplished by drawing attention to differences between off-the-shelf software products which directly affect user productivity, and by discussing some obvious enhancements to systems which can increase user productivity. However, the primary intention is to stimulate the use of at least some human performance measures in the competitive procurement of computer hardware and software in the Federal Government. Such measures would be part of the more general use of performance measurement technology in procurement actions. This is not to imply that human performance measures are less important in improving the use of existing systems, but it reflects the belief that change can be most readily and dramatically effected at the initial stages of systems development.

2. Some Human Factors in Computer Systems Use

Even a seemingly superficial cataloging of human factors in the use of computer systems reveals a general lack of consideration given to them in the systems acquisitions process. An attempt will be made to explain the importance of these oft-ignored factors--which do not have glaringly obvious price tags attached to them. Minimal attention will be given to the most-researched factors--i.e., factors relating to programmer performance in the use of computer systems. For such discussions, the reader is referred to [3].

2.1 Error Reporting

The most common experience, perhaps, for computer users of all skill levels, is having the system fail to perform a requested function, or to perform a different, unwanted function instead. A facetious observer might suggest that error messages were designed as though error detection and correction was to be a riddle to be solved by the user. Error reporting, whether system-caused, or user-caused, is an unwanted system perturbation which results in reduced efficiency at the human-machine interface. In the worst cases, the user is thrown back to a shelf of loose leaf notebooks or put into the expert's consultation queue before she can proceed with the work at hand. An example of this is the cryptic, "FAC REJECTED 400100001", from which the user is to infer that a file could not be found in system directories. Error reporting can also be excessive in its detail--users who are familiar with a system must put up with lengthy error diagnostics for errors perhaps caused by typo's. Nor should an attempt be made to "reach a happy medium"; there is no happy medium in the typical heterogeneous user environment. Multiple levels of detail should be available upon request. References to specific pages of written materials should be made at some level of depth if necessary. At the highest level, only the most necessary and self-explanatory information should be given. Various levels of error reporting could be turned off at the beginning of sessions, or turned off and on by the user at will.

The relationship between user actions and system reactions can be obscure. The user will frequently find herself saying, "now what does that message really mean?". And how frustrating to receive the consultant's answer, "Well, you left off a period on your file name; therefore the diagnostic has no meaning." Little wonder that new users remain bewildered as to how finite-state machines can appear to behave so illogically. There are instances of true ambiguity, of course, but the point is that there are far more than there need to be.

2.2 Systems Software Documentation

Most of the sources of problems with error reporting can be traced to systems software documentation practices. The purpose here is not to deal specifically with systems software design, but rather with its effect upon users. The extremely dynamic nature of systems software (changing several times a year in version if not more often) requires close monitoring of the relationship between new versions of software with new versions of documentation on that software. Because of the intrinsic interdependence of much software, simple replacement of pages here and there is not enough. The format of most systems documentation is also subject to criticism because of a clumsy organization which is based upon the internal software structure instead of the user perception(s) of system functions. Indexes are rarely adequate or complete, and are especially weak in terms of cross-references to related topics. Some ingenuity must be spent in the development of documentation benchmarks; e.g., take a naive user and ask her to perform a particular function "cold", with only the documentation available. Measurement of this time-to-criterion would obtain some revealing information about the design of the documentation. The documentation which would be measured should be based upon a representative job mix of the type of user and programmer behavior which is expected on the system. Normally, a broad spectrum of users would be required for a general-purpose system (general-purpose systems seems to get the most scrutiny).

Some industry standardization, perhaps using CODASYL as a prototype, regarding documentation would be highly desirable. The programmer or user who is required to move from system to system performing various functions in terms of access to data or software development or systems design is hard pressed to keep straight both differences and common capabilities. Transfer to training is most negative, unfortunately. Notably lacking are between-manual references (e.g., between the manuals for a compiler and operating system manuals), and annotated runstreams. The idea of "learning from example" is especially helpful in improving productivity in the use of a system; there is much to be said for the idea of getting the job done using a model, and then figuring out the conventions and the protocol later. Annotated runstreams appear in some training manuals, but for some reason, the managers of documentation staffs seem to be suffering from the delusion that reference manuals are novels or textbooks to be read chapter by chapter in a predetermined order, and examples play a secondary role in such exposition. Like the error reporting functions, documentation must be available at various depths of detail and theory. When one is looking for the fast answer to an acute problem, that is no time to discover, for instance, the theory behind the generalized syntax analyzer for the operating system's command processor.

Along these lines, there should be more effort to put together sections of documentation that are functionally related--not just theoretically or alphabetically related. For instance, when a cryptic message tells a user that she has used up all the disk space she allocated for a file, she wants to know what she can do about it, not all the theory behind how the granules, cylinders, tracks, sectors were estimated in the original allocation. Some of these interrelationships can be discerned from the error messages which the system issues, others from the common sequences of commands seen in annotated runstreams. Another example is the sequence of EDIT-COMPILE-LINK EDIT-EXECUTE-READ DATA-REPEAT. The reader of the EDITOR manual is not told how to compile a file created by the EDITOR. The reader of the compiler manual may not be told how to create a relocatable program. The reader of the LINKEDITOR manual may not be told how to execute a program which has been loaded, or how to associate files with the program that has just been created. A common structure for editor documentation would be a tremendous improvement.

2.3 Data Communications Facilities

The Teletype Model 33 user who is taken into the computer room and sees the systems console operating at 19.2 kbs may be prompted to say, astonished: "Hey, the computer <u>is</u> fast, after all!" For some of the functions involved in computer system use, higher baud rates (especially above 2400 bps) are highly desirable. While there may be some argument as to whether higher baud rates result in less "think time" for users and hence result in inefficient system use, most anyone would agree that, all other things being equal, higher baud rates are desirable.

Yet the matter of data communications design is not given a central role in systems design and implementation of a computer site. Perhaps much of the trend toward decentralization of computer facilities, toward smaller machines, is caused as much by the inability to get hard-wired high speed data communications service to larger, host computers as any other single reason. One might argue that to not have fast baud rates is to be cheated of the speed of the computer system, since the result that might have been computed in microseconds takes many seconds or minutes to be displayed on the screen. Similarly, adept, typewriter-wise users are frustrated by the inability of the computer to keep up with their nimble fingers darting to the "RETURN" or "TRANSMIT" key. Insufficient thought has been given to whether dedicated, conventional TTY (seemingly the most common based on CRT sales) or poll-and-select protocol is the most desirable, and what the trade-offs might be for a particular mix of users. Another example is the need to be able to utilize the editing features of the terminals available on the market today, and thereby reduce the editing load on the host system; i.e., line-by-line vs. screen-by-screen terminal-host transactions.

User needs and requirements, not terminal manufacturer or host default protocols should determine the type of data communications service to be used for any given case. Planning for diversity in the data communications environment of a large site seems to be the exception rather than the rule. There is an unfortunate temptation to standardize upon a protocol or two and then demand that the community of users conform to this arbitrarily determined standard.

2.4 Software Development Aids

There is scarcely a manufacturer who does not claim to have exceptional program development tools--debuggers, dump analyzers, subscript checkers, etc. These tools, and other software amenities, such as user program libraries (e.g., Programmer's Workbench under UNIX) become a necessity for large-scale software development projects. However, standardization of the functions or documentation of these programs has not occured, and therefore inadequate attention may be given to identifying what the necessary features of these programs may be for competitively selected systems. A careful examination of the system user mix and the past behavior of software development personnel would be the most reliable sources of information regarding such requirements.

In general, however, not a great deal of thought has gone into the development of such amenities, unless they were deemed necessary in the development of the operating system and the compilers (and then subsequently released to the public). For example, in spite of the encouragement of structured programming constructs, the user must keep track of "procedures" and individual routines within a physical "file", due to the rigid relationship between files and the utilities (such as the compiler itself) which must operate upon a single file--not a collection of files containing many procedures. Documentation for programs such as this, expect so far as they are self-documenting (and such a thing does not yet exist in a complete sense) must be developed and maintained totally external to the code itself, or else must comment within/among the code. Typical of the lack of understanding of the relationship between the process of program development and the operation of systems programs, is the term "pretty-print". This term implies that "pretty-printing", which refers to the indentation (and other "display" functions) of sections of code which are embedded within outer control loops, is a nicety, an amenity, rather than a necessity without which inefficient system use is inevitable.

The advent of protected fields, partitioned-screen editing, highlighting, blinking, etc., provides the hardware tools for implementing the capability to document programs while writing them, and subsequently, for documenting the use of these programs for their ultimate consumers, (at different levels of detail). As previously stated, the purpose here is not to discuss research into possible documentation standards, but to establish their relevance to human performance in the use of a computer system.

Other aids which may be considered as essential are structured programming "preprocessors" or compiler enhancements (e.g., Harris Corp.'s and UNIVAC's additions to FORTRAN) to support structured programming, and indirectly, structured design (which is even more critical). Some monitoring of the user's own behavior in system use--using many of the same software/hardware tools that are the stock in trade of the computer performance evaluator--would also provide much-needed feedback to users regarding resource utilization, optimization of code, and the conformance with design objectives.

2.5 Systems Software

The systems software, as commonality of parts become increasingly prevalent through the inroads made by OEM manufacturers, and as systems costs go down somewhat, will become the most important part of any computer system. Therefore, the human factors component of the systems software is the most important single aspect of the computer system. To fully discuss the human factors aspects of systems software would require more time and space than can be allotted here. Some areas indicative of the needs can be pointed out, however:

a. Common utility of systems software among all the compilers

b. Accessibility of systems utilities at the CALL level from all higher-level languages

c. Ability to make best use of hardware available (e.g., multileaved memory, cache memory, character or string manipulation hardware, etc.)

d. Software which tells the user what is going on (transparent to the user) if there is a need to know

e. Software's handling of contingencies, and user's control over how the contingencies are to be handled in different program environments

f. Simplicity of the job control language (as evidenced, e.g., by the ability of a user to "guess" which command performs a given function)

g. Generality of the JCL, i.e., the capability of the software to handle a variety of conditions, such as many file structures with the same processing logic while still remaining within a consistent framework of command structures and syntax

h. Language of the systems commands: how well they relate to their functions (do they actually do what they sound like they should do), how easily can they be remembered, how easily can they be associated with other commands which comprise part of a common logical sequence

i. System behavior when it aborts and recovers

j. "System memory" for events--e.g., remembering what the user has done, recovery of previous versions of files or programs, "undoing" changes to edited files, etc.

k. Capability to quickly create and intelligently examine "printouts" and data files on disk without waiting for listings to be printed, as part of test and evaluation phase of software development

l. Capabilities for applications-oriented software (commonly a need for transactional-oriented capabilities, for example)

m. Performance monitoring capability available to the user to aid in software development or in making intelligent use of systems or applications software (applicable to naive as well as sophisticated users if properly implemented)

n. Manipulation of "pieces" of data and programs rather than just entire physical files

o. Study of how long it takes to type in commands to get work done

p. Capability of creating files compatible with other machines

2.6 Accounting System

In the past, accounting for computer system utilization was regarded largely as a necessary evil. Today, in the "era of limits", accounting is more than a necessary function--it is a matter of survival to be accountable. Accountability and performance evaluation, fortunately, go hand in hand. However, particularly on smaller systems, and particularly when it comes to disk storage accounting, many computer systems are deficient in the accounting system which is provided. The accounting system needs to be well enough integrated into the system itself to: provide on-line, up-to-date accounting for usage within a timeframe of a given shift (less than 8 hours and preferably more often), and to account for usage across all resource dimensions by which systems and user software is utilized to do the computing. The oft-stated maxim that the most frequently used software should get the most attention in terms of optimization and documentation is a fine utterance, but this cannot be identified without the proper accounting software.

More importantly than this, however, is the fact that the accounting system for the computer utility in an organization is the most sensitive interface between the organization and the system. The accounting system can be the greatest source of annoyance to managers to contend with, the most distasteful software for the systems programmer to write or modify, the most dynamic data base on the system, an area sensitive to system crashes, and the most important in providing feedback to the user regarding efficiency of system use.

Ironically, it is this interface between the computer system and the organization which provides the greatest opportunity for performance evaluation in general. It represents the greatest source of control of user behavior on the system, while measuring system performance in terms of the organization's objectives--i.e., project-by-project accounting. Through the application of actual costs and weights in the cost algorithms, based upon true costs of the computer utility in the organization, one would expect a changing algorithm as capital equipment costs were recovered and ongoing communications, personnel, space, supplies, maintenance and power costs comprised an increased proportion of operating costs. However, because of the frequently very indirect relationship between actual systems support/use costs in the organization and the systems billing algorithm, the accounting system becomes a static entity, unresponsible to the organization and unresponsive to the recommendations derived from data which would be collected by systems performance analysts. Convention seems to dictate which systems parameters are assessed and which are not. The sensitive interrelationship between connect time, disk and tape storage, and channel utilization is disregarded in favor of facile algorithms which do not take into account the sequence of events which determine user behavior, the sequence of motions through which users go to accomplish typical tasks.

The general criticism may be made that the parameters utilized in most accounting systems are based upon the hardware costs/resources, rather than including software and labor costs. This is the reverse of the cost trends in computing.

2.7 Customization Capabilities

Programmers who design software systems whose primary consumers are naive users have special requirements of the systems software. In particular, they must present a hospitable facade to the user. The capability should exist which would permit "respectable" recovery from unexpected contingencies, which shields the naive user from distracting or meaningless system messages and acknowledgement, and which in particular permits the building of sophisticated command files or "runstreams". For some systems, it is desirable for the programmer to request accounting or performance information from the operating system at various stages of performing user-requested functions, in order to inform the user about potential or accumulated costs, or to assess the efficiency of the program's own techniques for processing. A typical application requiring such service is an on-line DBMS environment, with a generalized user language presenting the primary user-system interface. Another need arises from the desire to impose a consistency and a "sensibility" upon command structures and interactive dialog sequences which may not be present in the operating system's design. Once again, the structure of an operating system's interface to the usage consists of a myriad of undifferentiated elements which hang together only when viewed from the systems programmer's perspective upon them. Hence, only the capability for executing user programs to issue commands in the normal operating system format(s) can overcome some shortcomings.

2.8 Maintenance

Maintenance is a topic that also deserves a paper unto itself. The dearth of maintenance and maintainability performance evaluation at previous CPEUG conferences is reflective of the general attitude of the industry toward maintenance. Maintenance is viewed as a necessary evil. Technological developments (LSI) have been depended upon to achieve what common sense and a considerable body of literature on the subject has not achieved [4]. Some of the more neglected aspects of maintenance performance evaluation will be treated here; certainly a more exhaustive treatment is warranted.

a. Maintenance accountability. The process by which maintenance is undertaken remains largely understood except within certain academic and military applications circles. Federal contracts call for accounting for hours and parts involved in maintenance of Federal equipment, with penalties sometimes associated with not meeting minimum "up-time" requirements, but this is obviously a superficial treatment of a difficult subject. It would be a pleasant alternative, from the Government's standpoint, if all maintenance problems could be left to the vendors for solution, but this has not happened, and the Government's lead, with its huge investment in ADPE and enormous recurring costs for software and equipment maintenance, would be a much-needed stimulus.

One solution is to make more attractive the preferences given in procurement actions and contract negotiations to more maintainable equipment. At present, it would be difficult to establish that vendors in general even differ significantly from their maintenance records. Everyone seems to accept that disk and tape units require the most maintenance, followed by unit record and printing equipment, and the adoption of industry standard interfaces in some areas and the common use of components by the large OEM suppliers (Pertec, CDC, Calcomp, EMM, Memorex, Wangco, etc.) may have contributed to a homogeneous maintenance environment. However, even these hypotheses may be questioned, since access to data on software reliability and maintenance practices across vendors is also not readily forthcoming, and some Government initiative in that area as well seems called for.

b. Public knowledge of maintenance needs. The notion of maintenance as a behind-the-scenes necessary evil is a negative factor so far as user's understanding of potential problem areas and causes for failure is concerned. Some knowledge about the causes of failures and areas needing greatest maintenance could be mutually beneficial for users and maintenance technicians in reducing usage in troublesome areas where possible, and increasing sensitivity to intermittent failures which may be more difficult to track down. Clues to subtle systems problems may be shrugged off by users unaware of the situation.

c. Cost-effectiveness assessment of maintenance practice. The Government (or any other large-scale user for that matter) is faced with the troublesome problem of deciding whether on-call, dedicated, or redundant-component maintenance practices are required for a particular installation. The costs of these various alternatives can vary greatly, given the maintenance history of the system, the current cost of labor, and the cost of the most-likely-to-fail or most-catastrophic-if-failed components. Some attendance to the details of maintenance practices and costs could result in more informed decision-making about the most appropriate plan for providing repairs and replacements.

d. Logging of component histories. As board-level replacement and repair at the factory by unknown remote technicians and third party maintenance becomes more prevalent due to changing technology and design practices, automation of logging of component periodic maintenance and failure histories becomes essential. Manually kept logs are inadequate because automatic and timely component statistics are not available.

e. Automation of diagnosis and treatment procedures. Some of the procedures for diagnosis and treatment of computer system (hardware and software) problems are commonly known by better technicians, but even the best technician can easily overlook a logical possibility for system fault due to the complexity of the problem. Some automation of the diagnosis and treatment as well as intervening test and evaluation processes could be instigated even more than has been begun by some of the manufacturers such as DEC and BTI. For further research on this, see [5].

f. Life cycle costs for software/hardware maintenance. Increased attention in recent years to life cycle costs in ADPE have concentrated upon software maintenance costs more than hardware maintenance costs, and a more balanced perspective is needed. Also, the reduced cost of equipment causes the replacement with-new-machine option to become more attractive earlier in the life cycle. Then the problem is one of continuity between short-lived hardware systems. Five years is not a very long time for a project to exist, and for a considerable library of software to be accumulated, yet the hardware may be obsoleted and not supportable at the end of such a time. Such considerations affect the type of maintenance contracts or in-house capabilities the Government must provide.

g. Source code for mini and microcomputers. A deplorable practice exists in some ADP procurements-- those in which the Government does not receive the source code for systems routines. Certain critical routines are necessary for the Government's avowed multi-vendor system program. E.g., to interface some intelligent terminals with page printers of different manufacture, explicit knowledge and sometimes modification of the firmware controlling the serial printer interface are required. The increased use of firmware in computer systems, coupled with the longevity of much systems software which will be in use in the 1980's implies that greater attention be given to the matter.

h. Levels of detail for technicians. Some of the faults of systems documentation which were previously discussed hold true for the hardware documentation for computer systems. The effect is devastating when the technician is forced to muddle through schematics and theory of operation in order to search for the one small piece of information which she needs for troubleshooting. In order to solve this, some companies (such as Dustin Associates) have developed systems encompassing multiple levels of detail--so that all the same information remains available, but only at the relevant point in time in the diagnosis treatment or training process. Review of maintenance documentation in procurement, with "benchmarking" is required, with adoption of more stringent MILSTD regulations for major commercial procurements perhaps advisable. It is obvious that the vendors cannot be left to their own instincts on this matter, as the author's experience with 11 different manufacturers of terminals can attest. (Yet those writing procurement specifications may lack the technician's background needed to include such requirements.)

i. Accessibility and mechanical design factors. Despite some attention given to human factors in electronics systems in the human factors literature, there are unfortunate conditions which many technicians know exist. Components cannot be tested in place, and card extenders may not exist for them. Subassemblies may be mounted in backwards or upside down so that even cursory examination of cards cannot be performed without lengthly dismantling. A maze of interconnecting cabling may obscure components which must be troubleshooted. Inadequate cooling or mechanical factors may make working on some devices risky in terms of the damage which might accidently be done to the equipment. When parts fail and must be serviced, such factors should be brought to the attention of the contract monitor and/or supervisor for inclusion into maintenance documentation for the system component. Many of the factors discussed in a joint services study [6] have simply been ignored.

j. Federal Government-wide collection of failure data by component. An enormous untapped data base on MTBF and MTTR for computer components and sub-systems exists in the Government, which, if it were pooled, would be a tremendous asset to individuals responsible for planning new systems and evaluating potential equipment by vendors. This data base currently does not exist, nor is there a mechanism to facilitate or encourage the centralization of such information (except for a few large-scale military systems). This might be true, for instance, in the case of PDP 11's even though the number of PDP 11 systems in the Federal inventory may be in the thousands. Sharing of maintenance expertise within the Government, especially on off-the-shelf commercial equipment is also not facilitated though badly needed, especially in the design of distributed computing networks involving multiple systems of the same manufacture or configuration of components.

Related to this is the problem of non-standard nomenclatures for IC's and various electronic components, making it extremely difficult for Government technicians to cross-reference vendor-specific part numbers with industry-standard parts which the Government might stock in its own inventory. One spin-off of such a centralization of functions would be a Federal stocking system which would reflect the evidenced failures and the demand for spares. This demand currently is not visible because of the high percentage of contract maintenance, and the practice of obtaining parts and services from the immediate vendor rather than (in some cases) even the first manufacturer of the component who supplied the part(s) to the vendor.

k. Standardized procurement of maintenance aids & routines. Procurements of lesser dollar volume which are not part of larger systems typically do not include procurement of the necessary card extenders, spares, exercisers, and special alignment or test equipment which is associated with the equipment. The availability of such items to the vendor alone makes it impossible for the Government to work on the equipment itself--either to effect improvement, modifications or to permit interfacing with other equipment, and it makes it difficult to handle transitions from one maintenance contractor to another--should the original vendor no longer be awarded the contract for maintenance of the equipment. Commonplace delays in the Federal ordering process can be caused by waiting for maintenance aids, parts or technical data from the manufacturer, even though the items in question might have a nominal cost. Mandatory procurement of such items required for maintenance seems more than just a good idea.

2.9 Training Effectiveness

The matter of the training and educational materials made available to the user is one of considerable concern as turnover of personnel and increased travel and labor costs make training a major Government expense. To some extent, training materials should overlap with actual system documentation, but this appears not to be the case for reasons already discussed in connection with systems software documentation. Management of the computer utility includes certainly, some control over the training of new staff of new users, especially as the user population expands to include a greater number and diversity of people. Training effectiveness and its measurement is a speciality of my agency for the Navy, and therefore I know enough to say that this, too, is a subject deserving of separate study. But several areas of deficiency are blatant enough to justify some superficial criticism.

Until recently, many of the training classes held by the major manufacturers did not even include hands-on or even terminal-oriented exposure to systems. Rather, the emphasis has been upon theory of operation, with concentration upon the language and convention of the manufacturers' product line, rather than using some industry-wide language or concepts to explain system-specific notions. There has been little use of individualized instruction or even much involvement of professional educators or educational psychologists in the design of training programs or curricula for commercial computer systems. Certainly the presence or absence of such factors should make a difference in the acquisition process of evaluating vendor proposals.

Unfortunately, training is viewed as a one-time cost associated with initial system procurement, rather than an ongoing requirement which as much as any other single factor may determine system efficiency--poorly trained users presumably make inefficient use of computer and, indirectly, organizational resources. Training must keep abreast of new releases of operating systems, compilers and utilities, and the like. In other words, training, to be effective, must be contemporaneous with the system's state of the art, and must be available on an individualized basis. In the computer industry at least, the opportunity seems to exist, and also the economic incentive, (because of the system investment already made in training materiel support) to develop computer-assisted or computer-managed instructional modules to provide various levels of training assistance to the customer. Such features should become ranking factors in procurement decisions.

3. Incorporation of specifications into RFP's and technical evaluations

In order to implement some of these recommendations, it will be necessary to make some changes in existing procurement practices, especially as they relate to "benchmarking", "life cycle costing", and characterization of the "typical" job mix composition. This is a practicable suggestion which can be implemented, on a small scale at least, at the level of the technical individual responsible for writing specifications, and the contracting officer responsible for ultimately consummating a contract. However, for Federal Supply Schedule 74, 66 and 70 contracts, considerable negotiation will have already taken place; the incorporation of factors such as these insofar as they relate to ongoing support and provision of technical and maintenance information, must be done at that time and at that level, by GSA.

Encouragement of the inclusion of specifications such as those mentioned here can be facilitated by explicit mention of them in the various Federal regulations concerning the justification required for ADPE and data communications acquisition. More critical than this, however, for computer performance analysts that are the prime audience for this discussion, is the systematic inclusion of these variables in the assessment of computer system performance. This encompasses a theoretical as well as a practical domain in which user behavior in response to system hardware and software features is part of a feedback loop of events which ultimately, taken together, determine systems performance. Recommendations based upon isolated analysis of hardware or software alone will not necessarily result in increased systems efficiency if human factors in the use of computer systems are disregarded. This important area can also generate a host of stimulating new needs for systems software, documentation, hardware characteristics, and maintainability of systems which in a vacuum of systems modeling and simulation without user variables might never be considered. And, perhaps more importantly, the newly acquired broad population of computer users may be expected to become impatient with computer systems which were designed to be used most effectively by an imaginary, mid-level programmer with a clear understanding of how the system works; this is especially true since the system only begrudgingly yields clues as to how it can be used with the least amount of effort. Professionals in the computing industry may be forced to face the fact that the user view of computer systems as seemingly stubborn puzzles is a definite liability and an embarassment rather than an insider's source of amusement.

References

[1] Fitzsimmons, A. and Love, T., A Review and Evaluation of Software Science, ACM Computing Surveys, Vol. 10, No. 1, March 1978, pp. 3-18.

[2] Held, G., The Hidden Cost: Power, Data Communications, May 1978, pp. 41-52.

[3] Love, T., An Experimental Investigation of Program Structure on Program Understanding, in Wortman, D.B., ed. Proc. of ACM Conf. on Design for Reliable Software, Raleigh, N.C., March 28-30, 1977.

[4] Van Cott, H.P. & Kinkade, R.G., Eds., Human Engineering Guide To Equipment Design, Washington, D.C.: U.S. Govt. Printing Office, 1972.

[5] Electronics Test, debut issue, July/Aug 1978 (1050 Commonwealth Ave., Boston, MA 02215).

[6] Joint Services Symposium in reference 4.

CPEUG 78

TUTORIALS

COMPUTER SYSTEM SELECTION

WORKSHOP

S.A. Mamrak and P.D. Amer

The Ohio State University
Columbus, Ohio 43210

This workshop addresses the comparison and selection of computer systems. A methodology is presented which relies principally on the statistical analysis of measurement data obtained from the computer systems under study. One of the most important properties of the suggested methodology is that it incorporates confidence statements about the probability of having made correct decisions in the process of selecting the best computer. The workshop is divided into four sections:

1. Presentation of a Computer Selection Model

2. Brief Overview of Computer Comparison Experiments

3. Presentation of Ranking and Selection Procedures for Data Collection and Analysis

4. Example Application of the Selection Methodology

1. A COMPUTER SELECTION MODEL

An informal model of the computer selection process will focus the main components of the process and provide a framework for this workshop. The model is presented in Figure 1. It shows three phases of the selection process that lead progressively from the set of all computer systems under consideration to the isolation of the best one. Each phase involves the evaluation of the systems with respect to certain kinds of performance criteria. At each phase, those systems which fail to satisfy the respective criteria are eliminated. The kinds of performance criteria that are applicable are the topics of this section of the workshop.

2. COMPUTER COMPARISON EXPERIMENTS

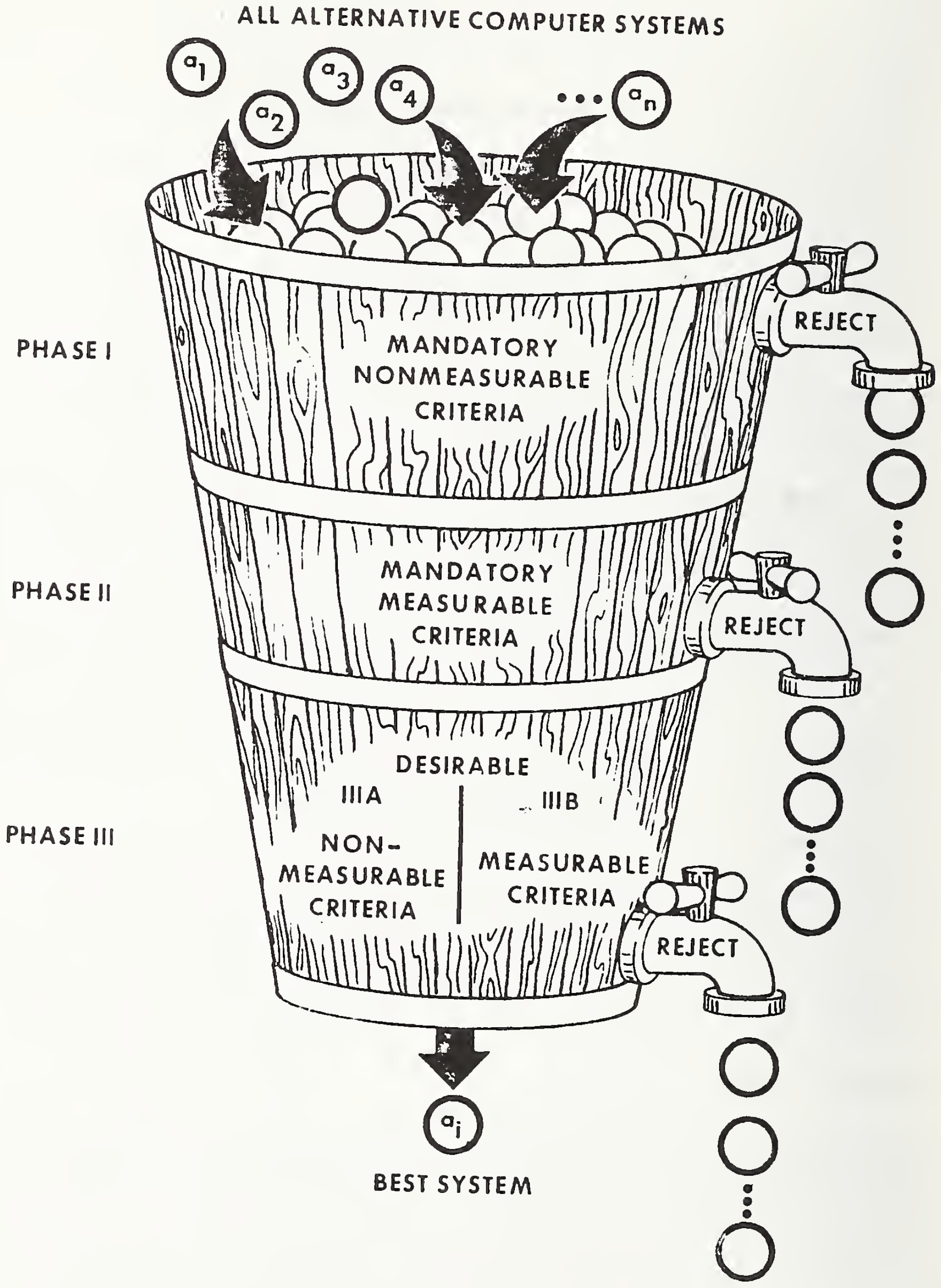
Some phases of the computer selection process involve running a representative workload on all of the alternative systems and collecting performance measurements on each system while it processes the test workload. The performance measurements are then analyzed and used as the basis for the selection of the best system. The measures generally chosen for comparison, and the constraint that the test workload be representative of the real workload, affect the type of experimental design and consequent data analysis that can be used.

The statistical properties of the data collected in computer selection experiments and their impact on experimental design are discussed in this section. The limitations on experimantal design introduced because of the representativeness requirement for test workloads are also discussed.

3. RANKING AND SELECTION PROCEDURES (GIB77)

A good experimental design is a critical component of any comparison methodology, since the efficiency of the data collection process and the validity of the data analysis depend on it. Statistical ranking and selection procedures provide an appropriate foundation on which to base computer selection experimental design. These procedures can be roughly characterized as following three lines of development: one set of procedures ranks systems by comparing sample means, one by comparing sample percentiles and one by comparing sample proportions. In all three cases, the procedures specify the number of data

Figure 1. Model of the Computer Selection Process



points which must be collected from each system in a comparison study in order to guarantee that the probability of a correct selection be greater than or equal to a predetermined value.

3.1 Selection Of Systems Better Than A Standard (AME78a,b)

Phase II of the computer selection process (see Figure 1) involves selecting those computer systems or systems which are better than a standard. An experimental design is required which leads to an analysis of the data that answers the question "Which services are at least as good as a prespecified standard?". The ranking and selection techniques which have been developed for selection better than a standard are not appropriate for computer selection when performance measurements are being compared at their mean or percentile values. These procedures make assumptions about the data that are clearly not justified in computer selection experiments. But, an appropriate procedure which does not require any assumption about the underlying form of the data's distribution does exist when proportions are the basis for the selection.

3.2 Selection Of The Best System (AME77b, MAM77)

Phase III of the computer selection process (see Figure 1) involves selecting the best computer system or service. An experimental design is required which leads to an analysis of the data that answers the question "Which system is the best one?". A full range of ranking and selection techniques exist for the selection of the best system when performance measurements are being compared at their mean, percentile or proportion values. The task of integrating the information provided by the methodology, if multiple performance criteria are the basis for selection, is discussed.

4.0 EXAMPLE APPLICATION (MAM78)

A large scale feasibility case study was done to evaluate the time and cost required to apply the computer selection methodology in an actual procurement environment. Four heterogeneous remote access time sharing systems were compared. The specifications for the case study and the experimental results are discussed in this section of the workshop.

References

AME78a Amer, P.D., "Experimental Design in Computer Comparison and Selection," Ph.D. Dissertation, Department of Computer and Information Science, The Ohio State University, December 1978.

AME78b Amer, P.D. and S.A. Mamrak, "Statistical Methods in Computer Performance Evaluation: A Binomial Approach to the Comparison Problem," Proceedings Computer Science and Statistics: Eleventh Annual Symposium on the Interface, Institute of Statistics, North Caroline State University, March 1978, pp. 314-319.

GIB77 Gibbons, J.D., I. Olkin and M. Sobel, Selecting and Ordering Populations: A New Statistical Methodology, John Wiley and Sons, New York, 1977.

MAM77 Mamrak, S.A. and P.A. DeRuyter, "Statistical Methods for Comparison of Computer Services," Computer, Vol. 10, No. 11, November 1977, pp. 32-39.

MAM78 Mamrak, S.A. and P.D. Amer, A Methodology for the Selection of Interactive Computer Services, National Bureau of Standards Special Publication, in preparation, 1978.

USE OF MODELS FOR CAPACITY PLANNING

J.P. BUZEN

BGS Systems, Inc.
Lincoln, Mass. 01773

1. Capacity Planning And Performance Calculation

Capacity planning problems usually arise as a result of some change in a system's external environment. For example, a teleprocessing workload may be growing at a rate of 3% per month, a new on-line application may be planned for the next fiscal year, or there may be plans to consolidate several data centers as part of a cost reduction effort.

In such cases the installation manager is faced with a number of options. He can: take no action and see if his system absorbs the new load; alter some aspect of his configuration (CPU, memory, I/O, etc.); alter some aspect of his software (dispatching priorities, mix of jobs in memory, etc.).

In order to deal with capacity planning problems, it is necessary to evaluate the impact of each of these options. That is, the installation manager must be able to determine how his system will perform in each case. To do this, it is necessary to solve the "performance calculation problem."

Essentially, the performance calculation problem can be stated as follows: given a description of a system's hardware, software and workloads, determine how the system will perform. Specifically, determine the systems's throughputs, response times, utilizations, and so on.

2. Benchmarking

There are two basic approaches to solving the performance calculation problem. The first approach is to run benchmark experments and measure system performance. That is, an actual workload is run on an actual system, and performance is measured directly.

Benchmarking may appear to be the ideal solution to the performance calculation problem, but it has a number of drawbacks when used in a capacity planning context. Note that the primary objective in this case is to determine how an actual system will perform under an actual workload. Usually they are a small sample (10 - 50 programs) from a workload that may contain hundreds of programs. The question is: is this sample truly representative, in the sense that the performance observed in the benchmark experiment will be comparable to the performance observed in the real system.

A subtle but equally important issue concerns the data being processed by the benchmark programs. The primary concern here is the distribution of the data across the various I/O devices in the configuration. The author has seen many cases where this distribution was grossly different from that in the real configuration, with respect to both the balance of loads across different devices and the distribution of seek times within individual devices.

Other problems associated with the use of benchmarks include the availability of the correct hardware configuration (sometimes an impossibility in capacity planning problems), the availability of application programs that have not yet been written, the appropriate representation of on-line workloads, and the handling of end effects.

3. Modeling

Models represent an alternative technique for dealing with the performance calculation problem. Essentially, a model is a computer program that receives, as input, a description of a system's hardware, software and workload. On the basis of this data, the model calculates what the performance of the system will be. Thus, a model is functionally similar to a benchmark, but quite different internally.

Most of the difficulties cited previously in the case of benchmarking are eliminated when a model is used. However, models raise a number of other concerns. The primary one is validity: does the model correctly calculate system performance, given that the description of the workload is accurate? Other issues in modeling concern the level of effort required to develop a model, the level of detail to include in a model, the availability of data to drive the model, and the choice of modeling technique (trace driven, stochastic simulation, analytic).

4. Forecasting

The performance calculation problem assumes that the analyst begins with a description of a system's hardware, software and workload. In capacity planning, the workload being investigated is often a future workload. Thus, workload forecasting represents a separate step that must be carried out before performance calculation can begin.

In most cases the load on a computer system is determined by external factors within the organization that the computer serves. These factors include the number of employees in the organization, the budget of the organization, the number of new projects started by the organization, the success of the organization's products and services, and so on.

These factors are clearly beyond the purview of the computer performance analyst. They should rightfully be supplied to the analyst by the management of the organization. However, it is then very important for the analyst to be able to translate these external factors into demands for the internal resources within the computer system. Techniques such as clustering and regression are particularly useful in this regard.

THE BASICS OF COMPUTER PERFORMANCE MANAGEMENT (CPM)

Capt. John C. Toole
AF Data Systems Design Center

This tutorial will provide participants with an overview of the terminology, techniques, concepts, and resource requirements which are necessary to successfully apply performance measures and systems. Limited experience in the field is assumed, and the tutorial is recommended for those just learning about Computer Performance Management (CPM) or people who have specialized in only one area. This tutorial will cover terminology, the tools and techniques of CPM, personnel requirements and training, and how to succeed with ADP management.

The tools and techniques discussed will include such topics as system accounting data, hardware and software monitoring, and prediction techniques (which includes modeling, simulation and benchmarking). Analysis techniques discussed include workload characterization and statistical methods. Within personnel requirements the CPM Team concept will be discussed.

A NEW DIMENSION TO THE DATA PROCESSING CENTER

Jame H. Garrett, Jr.

Value Computing Inc.
Cherry Hill, N.J. 08002

1. Introduction

The requirement for production control and scheduling of data processing data centers has expanded quite rapidly over the past several years.

This expanded requirement to apply better production control and scheduling techniques to the data processing operations area is a by-product of several factors.

* The primary factor is the continued growth in the number of new automated application systems which are being developed and implemented to service corporations' information needs.

* Secondarily, the continued growth of the transaction volume of existing applications has also contributed to the requirement for better production planning, scheduling and control.

* Questions concerning the types and capacities of equipment required, the number of work stations, the potential of distributing the workload and many other "what if" questions have also contributed to requirements for more sophisticated production planning and scheduling techniques.

The availability of more sophisticated equipment with larger, faster computing capabilities has made it possible for data processing management to better service the corporate information needs.

* With this more sophisticated, faster, and better configured equipment, the managements have had the ability to automate more of the existing applications by extending further into the user department's input and output streams. At the same time this new equipment has resulted in improving the data processing environment itself by utilizing more sophisticated on-line input and output oriented systems.

* This new, more sophisticated equipment has also facilitated the automation of application areas which previously were not considered to be cost effective or were marginal as far as their adaptability to previous data processing equipment.

The more sophisticated equipment, with expanded capabilities and a higher degree of service capability to the corporation and its end users, has brought with it potentially higher operating costs. Of equal importance, data processing has now extended itself, in many cases, into the main stream of the corporation's production of its goods and services. In many cases, computers control the entire production function.

2. Establishing Some Parameters

The first step in reviewing the requirements for Production Control within the data center is to establish a series of parameters from which the requirement may be examined.

Production control concepts are well established and have been refined to improve the control of many production functions. The classic functional breakdown of production control divides it into four major subfunctions. These are routing, scheduling, dispatching and follow up. Let's continue by further defining these terms.

* ROUTING - In the routing phase, a job or task is broken down into its manageable units that can be performed by a person

or machine. Each of the tasks is defined in detail and the order in which the tasks are to be performed is set down.

* SCHEDULING - The second phase of production control is the scheduling phase. In this phase, the element of time is imposed onto the work defined in the routing phase. First, work days within a calendar scheme are considered so that work can be scheduled by days in enough time to complete the product on time. After the work has been organized by day, more detailed time schedules are developed to give an hourly time dimension to tasks within idividual work centers or departments, to insure an organized work plan through the entire pro duction process. By combining the layout of work in the routing phase with the organization of the work by day and time in the scheduling phase, a fundamental organization of the work is accomplished so the work can then be dispatched and follow up performed.

* DISPATCHING - With the work now scheduled through the individual work centers, the work can now be dispatched. Dispatching is the third phase of production control and consists of the movement of the work from one scheduled area to another with instructions at each work center provided to insure that the proper steps are performed to complete the task.

* FOLLOW UP - The final phase of production control consists of follow up. In the follow up phase, the work is monitored to insure that the tasks follow the prescribed routing and scheduling plans. Further historical records are kept and acutal performance over time is compared to schedules so that new and improved schedules can be developed.

These four phases of production control have been existent in data processing operations and data centers for a number of years. They have appeared in varying degrees predicated on their requirements within a specific organization and its particular position along a time line relating to level of sophistication of the equipment installed, the volume of work to be processed and the variables affecting the work to be processed.

These techniques, when examined, form the basis for beginning to establish a production environment which is built around the scheduling system.

3. The "Key" Production Control Function-Scheduling

Any one who has ever had actual experience with production control knows that the heart of the production control system lies in the proper scheduling of the work. What value if the proper routing, dispatching or follow up if the work has been scheduled on the wrong day or in a haphazard fashion through work centers with little regard to the resources that can be applied at each work center?

In order to perform the scheduling function within the production control process, several factors are key.

* Considerable amounts of historical information are required concerning various jobs and job mixes.
* The sequence in which jobs are to be processed is also a very key factor.
* A thorough understanding, statement and examination of the resources available to process the work must also be prepared.

These three elements can vary quite radically from day to day, or hour to hour, within a data center operation.

As a result, specific techniques must be applied when dealing with the data processing evironment and the production control function.

The data processing management has been provided, through the operating system facilities, some very unique production control facilities. These facilities are not readily available on the part of other managements required to manage and plan production.

One of the facilities of the type we are referring to is the ability to vary the number of initiators which can process work. This is the equivalent of varying the number of machine tools available to a production manager in a manufacturing facility.

The manufacturing management very rarely has the ability to change the numbers of machine tools. At the same time, with the flexibility offered within the operating system, we can also vary the capacities of each one of our initiators or equivalent machine tools.

We can, by expanding or contracting core or assigning faster or slower input/output devices, change the amount of production which can occur or the amount of time it it takes to produce a certain quantity of work.

Again, conventional manufacturing production management does not have this alternative of varying their particular productive capacities without a considerable amount of activity and, in some cases, large investments in new equipment.

We, of course, through the operating system facilities and conventions, have another facility which is to change job class and/or priority as it relates to our processing streams on a dynmaic basis.

These combinations of facilties can be viewed either as an asset or a liability relating to the production function.

They become an asset or liability based on the level of scheduling expertise which is available.

With the ability to schedule the resources and make adjustments to this schedule on a dynamic basis, we can leverage in an optimum manner the flexibility of the equipment resources, operating system and support organizations available.

Of course, the optimum schedule may not be met at all times. However, in the actual operating environment, the degree of attainment of the optimum schedule is going to be based on the amount of flexibility that is offered in the scheduling technique or algorithm which is utilized.

The scheduling algorithm is required to provide management the ability to examine and consider the effects of multiple configurations or production processes given a certain level of work to be accomplished and a certain level of resource; i.e., input/output devices, core, numbers of initiators, etc.

The potential value of the entire production control planning process is significantly reduced if this facility to derive answers to the "what if" questions is not available.

The "what if" production control planning process is most frequently built around a scheduling formula or algorithm.

This scheduling algorithm provides management facilities to develop a series of iterations by varying the production elements and workload in the algorithm. The result is management can examine the effect and optimize its plans.

4. The Value Computing Automated Production Control System

4.1 General

Since 1969, Value Computing has been developing an automated production control and scheduling system for the data center operations. Value Computing's actual experience in developing automated data center production control systems for many data center installations has led to the establishment of the shceduling algorithm as the cornerstone of the data center production planning process.

Value's scheduling algorithms are Phase II of the classic production control process and produce detailed but totally realistic schedules and standards of performance.

The Value Computing Production Control and Scheduling System has been applied to single and multiple machine environments. The system has significantly contributed to improving MP type operating environments.

The system has been developed over a nine year period of time to accomodate the wide variety of work stations and environments found in various types of data centers.

4.2 The Elements of the System

Value's system mirrors the traditional production control process and has been implemented with the following components:

1. The Data Base - It is in the Value data base that the individual jobs are arrayed into networks that thread themselves on machines as well as between individual work centers. It is, therefore, in the data base that the routing phase of production control takes place.
2. The Scheduling System - The Value Data Center Scheduling System is the vehicle used to produce the hour by hour work plans for each work center in the data center. The algorithms in the scheduling system perform the scheduling phase of the traditional production control.
3. JCL Input Sybsystem - After the work plan for the computers has been developed by the scheduling system, the APOLLO JCL Input Subsystem is

activated to submit the JCL and control cards to the CPU in plan sequence. This process eliminates the handling of cards altogether and also allows temporary changes to be made on line. APOLLO performs the dispatching phase of production control.

4. Status and Revision Subsystem - The crucial function of followup which is the fourth element of production control, is performed by the Status and Revision Subsystem. This module reports in a real-time manner job and network status and monitors job sequenced produced by the scheduling system. When jobs are run out of sequence, work center personnel are advised on-line so corrective action can be taken.

4.3 Dynamic Job Release

Total machine control is provided in the system by linking the APOLLO and STARS systems together in such a way that APOLLO will automatically release jobs for execution on the CPU without manual intervention.

Thus, for the first time total data center production control with dynmaic job release is provided to computer operations so operators are no longer required to perform many clerical functions but, in fact, are freed to monitor for the exceptions in processing.

5. Summary

With the VCI Automated Production Control System, realistic plans for the data center are provided along with on-line JCL submission and job tracking. The system fully implements the four basic elements of production control; routing, scheduling, dispatching and follow up.

SAS - A UNIFIED LANGUAGE FOR CPE

William R. Gjertsen
SAS Institute, Inc.

Robert M. Gaddy
Duke Power Company

The Statistical Analysis System (SAS) is a unified system for data analysis. Within a single job step, the user can retrieve and manage performance data; generate plots, charts and summary tables; do statistical modelling and forecasting; and produce custom-formatted reports. SAS combines the power of a file management and retrieval system with the simplicity of one-statement calls to analysis procedures. It can be used for reducing data from software monitors, hardware monitors, data base log tapes, job accounting and security information systems, and many others.

The tutorial will first sketch how SAS can be a vital component in any implementation of a Performance Management System (PMS). Secondly, we will illustrate its flexibility, power and ease-of-use as a higher level programming language. Lastly, we will examine some current performance areas of interest at Duke Power Company and indicate the utility of SAS in a corporate setting.

1. Introduction

A PMS can be broadly defined as the implementation of management and control over the essential performance data bases within an organization. Control is needed to identify problems while they are still manageable, allow for timely resolution of those problems and give management the opportunity to direct rather than react.

Because raw performance data is voluminous, dispersed and awkward to process, an efficient method to retrieve, manipulate, store, archive, report and analyze the data is essential. To accomplish these PMS-oriented goals a Computer Performance Evaluation and Control (CPEC) system can be realized with SAS. The concept has been described by Morino[1] and implemented with SAS code by Merrill.[2,3] A CPEC as depicted by Morino is shown in Figure 1.

This set of data sources could be expanded to include more of the PMS concept by incorporating additional measurement data on hardware and media reliability, data base management system activity, communication activity and security information as additional inputs to SAS. The CPEC would:

- o Provide management with short-term control (daily).

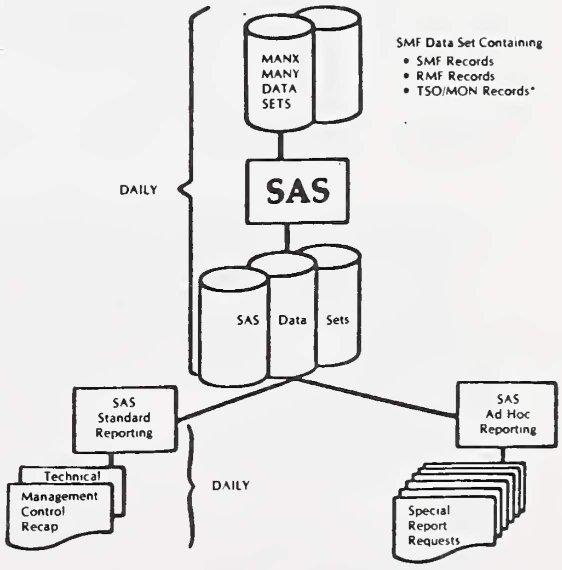


Figure 1: A CPEC System

- o Report on system availability, service, load, peak loads, and effectiveness of the TSO-Batch environment.
- o Apply "Control Interval" analysis for assessing how pre-determined objectives are being attained or missed.
- o Employ "Exception Reporting" to highlight and identify incidents requiring action.
- o Generate "overview" management control reports.
- o Provide generalized "Ad Hoc" reporting to supplement and support the control reports and problem resolution process.

2. An Exception Report

One type of report that management can act upon is an exception report. SAS has the necessary tools to extract selected record types, read many internal formats (EBCDIC, binary, packed decimal, date and time formats etc.) and handle repeated and variably positioned fields that so often occur in SMF and RMF records. Here is the complete code to extract and produce an exception report from SMF type 4 records:

```
// EXEC SAS
//SAS.SMFTAPE  DD  DSN=SYS1.MANX,DISP=OLD
//SAS.SYSIN  DD *

DATA; INFILE SMFTAPE;
      INPUT @2 REC_TYPE PIB1.@; IF REC_TYPE=4;
      INPUT @55 PGM_NAME $8. @103 DEVLEN PIB2.
      @103 +DEVLEN +1 CPUTIME PIB3.2;
PROC CHART;
  HBAR PGM_NAME / SUMVAR=CPUTIME TYPE=MEAN;
  TITLE MEAN CPUTIME (SECONDS) BY PROGRAM NAME;
```

3. SMF-RMF Reduction

Data reduction on RMF (SMF-MF1 records, RMF1, RMF2) records is easily handled by SAS. An outline of code together with a specific plot of type 72 workload records is given below. We are indebted to Terry Flynn for supplying the code for this example. The overlay plot for TSO response time (PGN=2) vs. time of day for different service unit types is given in Figure 2.

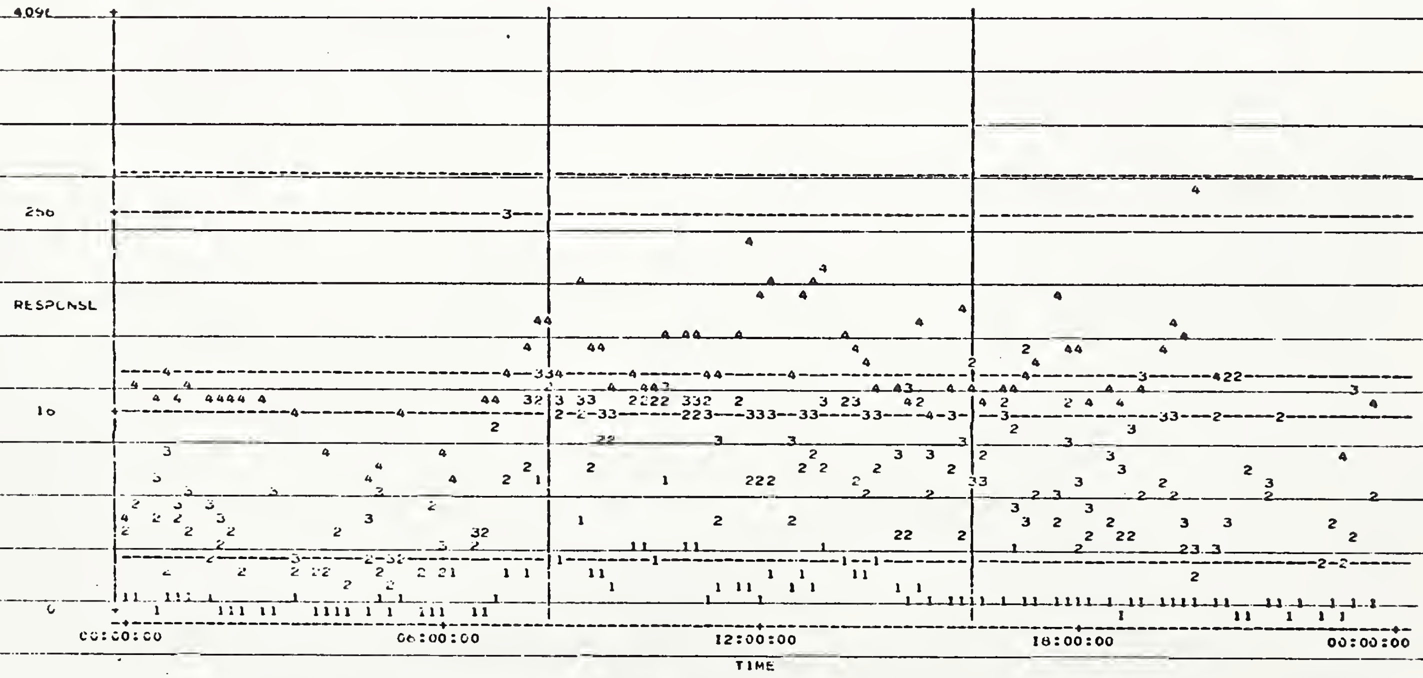
```
// EXEC SAS
//INP DD DSN=SYS1.MANX,DISP=SHR
//SYSIN DD*

DATA S70 (KEEP= CPU time field variables)
     S71 (KEEP= Paging variables)
     S72 (KEEP=DATE TIME DURATION PGN PERIOD TRANS
         ACTIVE TRANSERV ELAPSED RESIDENT RESPONSE)
     S73 (KEEP= Channel Record variables)
     S74 (KEEP= Device information variables);
 INFILE INP; INPUT @2 REC_ID PIB1. @;
 IF REC_ID=70 THEN GOTO _70;
 IF REC_ID=71 THEN GOTO _71;
 IF REC_ID=72 THEN GOTO _72;
 IF REC_ID=73 THEN GOTO _73;
 IF REC_ID=74 THEN GOTO _74;
DONE: DELETE;

_70: <INPUT Type 70 Records>
         :
_71: <INPUT Type 71 Records>
         :

_72: *INPUT TYPE 72 RECORDS; POINTER=15;
 INPUT @ POINTER COM_SIZE  PIB2. TIME PD4.
        DATE PD4. DURATION PD4.3 +2  PGN PIB2.@;
 IF PGN ¬= 1 & PGN  ¬= 2 THEN GO TO DONE;
 DURATION=60*FLOOR(DURATION/100)+MOD(DURATION,100);
 TIME=3600*FLOOR(TIME/10000)+
      60*MOD(FLOOR(TIME/100),100)+
       MOD(TIME,100);
 DATE=DATEJUL(DATE); POINTER=POINTER + COM_SIZE;
 INPUT @ POINTER WC_SIZE PIB2. NO_PG PIB2.
        PG_SIZE PIB2. +2 IPS $8. @;
 POINTER=POINTER + WC_SIZE;
 PERIOD=1;

_72_1:  INPUT @ POINTFR TRANS PIB4. ACTIVE PIB4.
                TRANSERV PIB4. ELAPSED PIB4.
                +16  RESIDENT PIB4. @;
 ACTIVE=ACTIVE * 1024/1000000;
 ELAPSED=ELAPSED* 1024/1000000;
 RESIDENT=RESIDENT*1024/1000000;
 IF TRANS  ¬=0 THEN RESPONSE=ELAPSED/TRANS;
 OUTPUT S72;     IF NO_PG=PERIOD THEN GO TO DONE;
 PERIOD=PERIOD+1; POINTER=POINTER+PG_SIZE;
 GO TO _72_1;

_73: <INPUT Type 73 Records>
         :
_74: <INPUT Type 74 Records>
         :
<Analyses for Type 70 & 71 Records>
         :
*--------------------------*
*                          *
* TYPE 72 WORKLOAD ANALYSES*
*                          *
*--------------------------;

PROC SORT DATA=S72; BY DATE PGN;
PROC PLOT; BY DATE PGN;
  PLOT RESPONSE*TIME=PERIOD/VREF=4 16 256 1024
       HAXIS=0 21600 43200 64800 86400
       VAXIS=0 16 256 4096
       HREF=2800 57600;
  FORMAT DATE MMDDYY8. TIME TIME.;
    :
<Further Type 72 Record Analyses>
    :
<Analyses for Type 73 & 74 Records>
    :
*--------------------------------*
*                                *
* ANALYSIS OF MERGED RMF RECORDS *
*                                *
*--------------------------------;

DATA ALL; MERGE S70 S71 S72 S73 S74; BY DATE TIME;

<Selection of Records, Creation of new variables>
         :
<Analyses on the merged RMF Records>
```

Figure 3 shows the generated workload plots. As this and the previous example illustrate, SAS differs from other CPE package programs in that the user defines the reports and graphs he needs by selecting the variables, SAS procedures, and combinations of data sets that are relevant to the task at hand. It is in this sense that SAS is a versatile language for CPE.

```
           MEAN CPUTIME (SECONDS) BY PROGRAM NAME                            1
                                       19:57 MONDAY, APRIL 3, 1978

                     BAR CHART OF MEANS

PGM_NAME                                   FREQ     CPUTIME
                                                       MEAN
           |
ASMGASM    |*****                              8    1Ø2.7712
A2Ø9Ø1ØA   |*                                  1     1Ø.47ØØ
A2Ø9Ø1ØB   |*                                  1     11.Ø9ØØ
A2Ø9Ø1ØD   |*********                          1    18Ø.Ø1ØØ
COSTTURB   |*******                            2    139.8Ø5Ø
DETAILS    |*                                  1     26.2ØØØ
FDR        |***                               12     56.6283
HMASMP     |*                                  2     1Ø.55ØØ
IEBCOPY    |*                                  4     16.4775
IEBGENER   |                                  44      2.8436
IEFBR14    |                                  37      Ø.Ø214
IEHIOSUP   |                                   2      4.575Ø
IEHLIST    |*                                  2     21.36ØØ
IEHPROGM   |*                                 16     19.8581
IERRCOØØ   |                                  33      2.9627
IEWL       |                                  27      2.Ø326
IFCEREPØ   |**                                12     38.7ØØ8
IMASPZAP   |                                   4      1.85ØØ
N38818RA   |                                   1      9.63ØØ
N38818RB   |*                                  2     19.975Ø
N38818RD   |****                               2     79.62ØØ
N38818RE   |*****                              1    1Ø6.11ØØ
N541PSB    |*******************************    3    624.36ØØ
N541Ø1ST   |**                                 1     36.86ØØ
N556Ø6B    |****************                   2    323.345Ø
PGM=*.DD   |*                                 12     12.3158
TMSAUDIT   |                                   1      2.Ø8ØØ
TMSBINQ    |                                   1      1.85ØØ
TMSCLEAN   |*                                  1     21.Ø6ØØ
TMSCOPY    |**                                 1     38.84ØØ
TMSCTLG    |**                                 1     32.87ØØ
TMSCYCLE   |**                                 1     33.53ØØ
TMSEXPDT   |*                                  1     11.14ØØ
TMSRPT2A   |*                                  1     29.84ØØ
WBD5Ø1FA   |                                   3      4.2867
WBD5Ø1FB   |                                   1      Ø.89ØØ
WBD5Ø5NA   |*                                  3     26.7967
WBD5Ø5QA   |                                   1      8.48ØØ
WBD5Ø5RA   |***                                1     53.88ØØ
WBD5Ø5SA   |***                                1     57.71ØØ
WBD5Ø5TA   |***                                1     55.18ØØ
WCCATUPD   |*****                              1    1Ø8.43ØØ
WSØ673ØA   |                                   1      Ø.5ØØØ
WSØ72ØPZ   |**********                         3    19Ø.73ØØ
XLFF       |                                   3      Ø.2933
           -----+-----+-----+-----+-----+-----+--
               1ØØ  2ØØ  3ØØ  4ØØ  5ØØ  6ØØ

                     CPUTIME MEAN
```

Figure 2. An Exception Report

```
                                   SMF REDUCTION                          11:02 THURSDAY, FEBRUARY 16, 1978     9
                                  CPU UTILIZATION
                                DATE=02/15/78   PGN=2

                       PLOT OF RESPONSE*TIME     SYMBOL IS VALUE OF PERIOD
```



```
NOTE:     4 OBS HAD MISSING VALUES OR WERE OUT OF RANGE     55 OBS HIDDEN
```

Figure 3. Overlay Plots of TSO Response vs. Time of Day

By Performance Group Number

4. Kiviat Star Charts

Another type of report currently in vogue is the Kiviat graph which attempts to highlight "good" and "bad" system usage values on alternate axes of a star chart.

Suppose we have stored daily utilization data for the months of July through December, and suppose as part of a year-end recap report, management requires an ad-hoc look at the system's utilization profile by month by showing KIVIAT charts of each month's mean profile statistics. This job first requires reducing daily utilization to overall monthly utilization and then producing the star charts. The SAS code and a plot of July's utilization follow.

```
// EXEC SAS
//M DD DSN=SYSUTILZ.MONTHLY,DISP=SHR
//SYSIN DD *
  DATA (KEEP=ANYCHBUS CPUWAIT CPUBUSY SPVST
        PROBST CPUONLY CPUCHOV CHANONLY MONTH);
   SET M.JUL M.AUG M.SEP M.OCT M.NOV M.DEC;
  PROC SORT; BY MONTH;
  PROC MEANS NOPRINT MEAN; BY MONTH; OUTPUT OUT=ONE
   MEAN=MCHBUSY MWAIT MBUSY MSPVST
        MPROBST MONLY MCHOV MCHONLY;
PROC FORMAT; *SET UP VALUE  LABELS;
 VALUE KIV
  1='ANY CHAN BUSY'    2='CPU WAITING'
  3='CPU ACTIVE'       4='SUPERVIS STATE'
  5='PROBLEM STATE'    6='CPU ONLY'
  7='CPU CHAN OVLAP'   8='CHANNEL ONLY';
DATA TWO; SET ONE;
 C=1; X=MCHBUSY  ; OUTPUT; *SET UP A SEPARATE OBS;
 C=2; X=MWAIT    ; OUTPUT; *FOR EACH POINT;
 C=3; X=MBUSY    ; OUTPUT; *OF THE STAR;
 C=4; X=MSPVST   ; OUTPUT;
 C=5; X=MPROBST  ; OUTPUT;
 C=6; X=MONLY    ; OUTPUT;
 C=7; X=MCHOV    ; OUTPUT;
 C=8; X=MCHONLY  ; OUTPUT;
 FORMAT C KIV.
PROC CHART; BY MONTH;
 STAR C/SUMVAR=X TYPE=SUM DISCRETE AXIS=0 1;
```

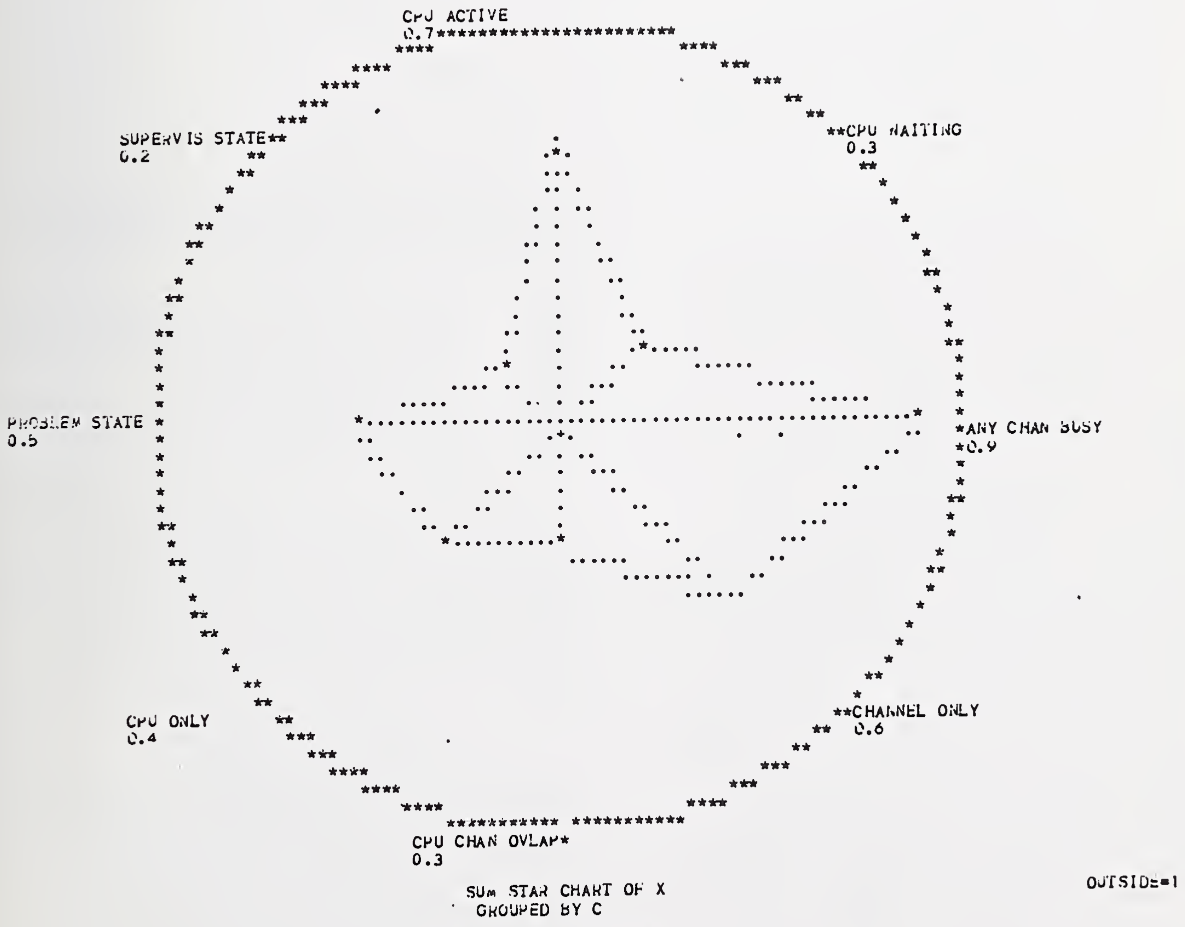


Figure 4. A Kiviat Star Chart

5. SAS in a Corporate Setting

Duke Power Company is a large electric utility company serving the Carolinas. Among public utilities nationally it ranks twelfth in overall assets and tenth in 1977 net income.* Although SAS is used there for load projection, environmental and biological analyses, and as a system utility for job accounting, this paper only addresses the aspect of performance tuning at Duke Power. This paper should be viewed as expository and should not be interpretted as recommendation or testimonial for SAS by Duke Power.

Duke Power has at its disposal an Amdahl V6-2 with 6 Meg, an IBM 370/165 with 3 Meg and an IBM 370/158 with 3 Meg. There are 14 tape drives and 56 Itel 7300-11 disk drives. The disks are shared between all three systems. At present MVT is the production operating system on all three machines and MVS is being tested on the 158. Cut-over to MVS is planned for the fall of 1978.

In addition, for performance analysis under MVT, Duke Power uses a COMTEN 8016 hardware monitor, Boole and Baggage's TSA and PPE products, SUPERMON, and a TSO Trace data reduction program. Morino Associate's TSO/MON is also currently being evaluated.

Current evaluation concerns are:

1. How can disks, channels, and channel controllers be tuned for optimum CPU utilization, TSO response time and batch turnaround?

2. What will be the impact on utilization and response time (TSO and batch) in migrating from MVT to MVS?

6. Tukey Schematic Plots

After the very first look at the DASD problem, it was conjectured that channel 3 was too heavily loaded. It was decided to first take a look at the percentage of the time channel 3 is busy (CH3BUSY) vs the percentage of the time the CPU is busy (CPUBUSY). But when you do a scatter plot on thousands of observations you are left with a useless cloud of points.

Another approach is to use an exploratory data analysis technique proposed by Tukey^4 for large batches of data (like typical raw performance data). One such schematic SPLOT or bar-and-whisker plot is depicted in Figure 5.

If you group CPUBUSY percentage into 10% ranges and do schematic plots of CH3BUSY within each CPUBUSY grouping you get a coherent picture of what's going on. In fact, Figure 6 shows that no matter how busy the CPU is, CH3BUSY puts out a maximum mean of 47%, a median of 50% and a maximum midrange of about 38 to 58% and this occurs whenever the CPU is 40 to 90% busy. Also when CPUBUSY is 90% or better CH3BUSY dips down to a mean of 34%. Since CPUBUSY is the response variable of interest it was decided to plot the dual of Figure 6 in Figure 7 (namely schematic plots of CPUBUSY within each 10% grouping of CH3BUSY). From Figure 7 we notice that CPU utilization peaks when the demands on CH3BUSY are 60 to 70% and thereafter diminishes as channel 3 gets busier. Since Figure 6 tells us that we are above this range about 11% of the time for a wide range of CPUBUSY (20 to 100%) it made sense to take some of the load off channel 3 and bring CH3BUSY out of the land of diminishing returns.

The vary-path command was in fact used to take some load off channel 3 and an increase i- CPU utilization was noted. In fact, the way to distribute the load is to minimize the time that any of the channels taken in unison are in their region of diminishing returns. One-way SPLOTS or adjacent SPLOTS as in Figure 6 and 7 can provide directions and insights for tuning and are a valuable aid in "picturing" large batches of performance data.

*Fortune Magazine, May 1978

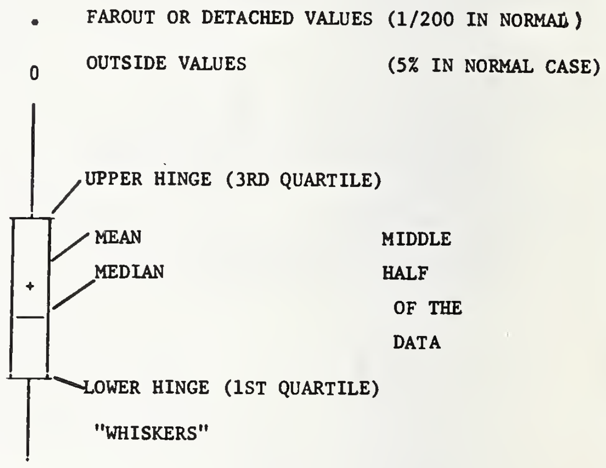

FAROUT OR DETACHED VALUES (1/200 IN NORMAL)
OUTSIDE VALUES (5% IN NORMAL CASE)
0
UPPER HINGE (3RD QUARTILE)
MEAN
MEDIAN
+
MIDDLE HALF OF THE DATA
LOWER HINGE (1ST QUARTILE)
"WHISKERS"

Figure 5. A Schematic Plot (SPLOT)

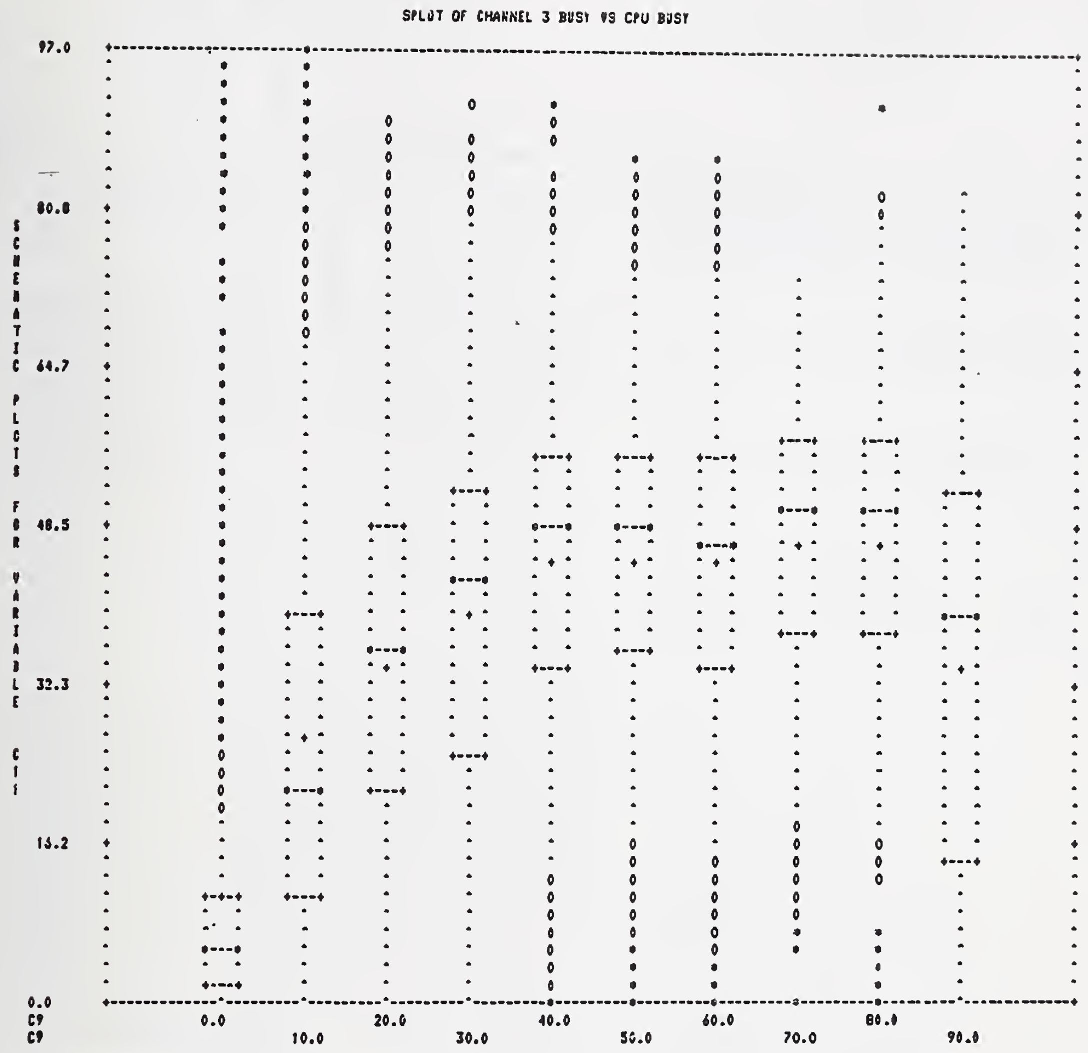
The schematic plots shown in Figures 6 and 7 on 10,073 observations can each be generated in less than 3 minutes by using SAS under TSO. Here are the statements for Figure 7.

```
00010     DATA; SET SASSPACE.S410 (KEEP=C9 C11);
00020     TITLE SPLOT OF CPU BUSY VS CHANNEL 3 BUSY;
00030     OPTIONS TLS=130;
00040     IF C11=100 THEN C11=99.9;
00050     C11=10*INT(C11/10);  *GROUP C11=CH3BUSY;
00060     PROC SORT; BY C11;
00070     PROC SPLOT; CLASSES C11; VAR C9;
00080     RUN;
```

Figure 6 is produced with the same code just interchanging C9 and C11 and changing the title statement. The raw data for this example was a week's worth of 5 second interval data measured by the COMTEN hardware monitor and put on tape. SAS then extracted the data from the tape, reduced the percentage to minute intervals, and created SAS data set S410 prior to producing the schematic plots.

7. Stepwise Regression

We can only hint at approaches to the initial tuning problems that will arise in the MVT to MVS migration. However, one technique that should prove valuable for detective work is partial correlation or equivalently stepwise regression. This is a variable selection technique which can give answers to questions like "What variables are affecting TSO response time?" This is a useful question to ask, especially if there are some dissapointing surprises after a migration or system configuration change. Using TSO/MON records



```
NOTE: SAS INSTITUTE INC., P.O. BOX 10066, RALEIGH, N.C. 27605
READY
```

Figure 6. Schematic Plots of Percent Channel 3 Busy By Percent CPU Busy in 10% Bands

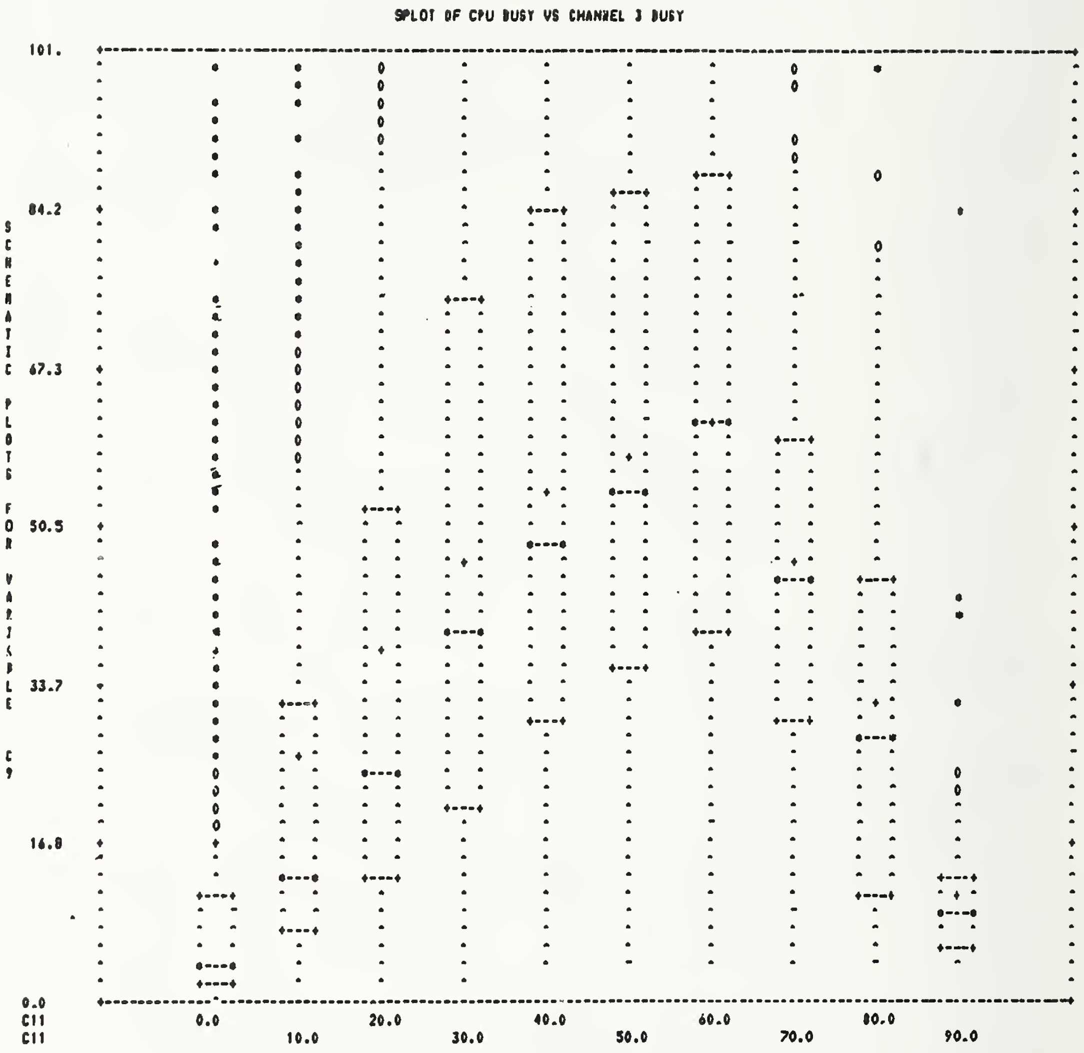


Figure 7. Schematic Plots of Percent CPU Busy By Percent Channel 3 Busy in 10% Bands

merged with RMF the SAS user could create a data set with appropriate TSO system and command information, CPU workload, paging, channel and device information. If concurrent batch activity is needed, SMF could also be included. After creating the data set of fields merged by time, you could do stepwise regression with a statement such as:

```
 PROC STEPWISE; MODEL Y=X1-X30/MAXR  STOP=6
                                INCLUDE=3;
Where Y= Total Response Time (Average of all re-
         sponse events).
     X1= Time period ( The interval  of activity).

     X2= Max Users (The maximum # of users logged
         on).
     X3= Perf Group (The TSO Performance Group
         Number).
     X4= Swap Count (The number of swap outs done
         during the interval).
     X5= SIO Count (The number of successful start
         I/O's).
     X6= VIO Page (Available VIO local Page Data
      .  Set Slots).
      .
      .
      . <Any set of possibly good regressors>
      .
    X30=
```

The MAXR option selects variables in a forward stepwise manner, but chooses from a larger selection of possible regressions than does the standard stepwise selection procedure.[5] The STOP=6 option tells SAS to select the best 6 and then stop and INCLUDE=3 forces the first three regressors into the model.

References

1. Morino, M.M., "SAS - The Necessary Utility for Computer Performance Evaluation and Control (CPEC)," Proceedings of the Third Annual Conference of the SAS Users Group International, SAS Institute Inc., Raleigh, (1978).

2. Merrill H. W., "Statistical Analysis of SMF Performance Data," SHARE Proceedings, Share Inc., New York (1974).

3. Merrill H. W., "Using SAS to Tune MVS," Proceedings of the Third Annual Conference of the SAS Users Group International, SAS Institute Inc., Raleigh (1978).

4. Tukey J. W., Exploratory Data Analysis, Addison-Wesley, Reading, Massachusetts (1977).

5. Barr.A. J., Goodnight J. H., Sall J. P. and Helwig J. T., A Users Guide to SAS76, Raleigh, North Carolina (1976).

CPEUG 78

APPENDIX

CPEUG 78 PROGRAM
TRACK A

Tuesday, October 24

WELCOME

Richard F. Dunlavey
CPEUG Chairman
National Bureau of Standards
Washington, DC

Robert W. Martin
Regional Commissioner
GSA/ADTS Region No. 1
Boston, MA

KEYNOTE ADDRESS

Frank J. Carr
Commissioner Automated Data and
Telecommunications Service (ADTS)
General Services Administration
Washington, DC

PROGRAM OVERVIEW

Terry W. Potter
CPEUG Program Chairman
Digital Equipment Corporation
Maynard, MA

Arnold Johnson
CPEUG Co-Chairman
Department of the Navy/FCCTS
Washington, DC

TECHNICAL ORIENTATION

FRAMEWORK FOR COMPUTER SYSTEM STUDY
R. A. Orchard
Bell Telephone Labs.
Piscataway, NJ

SELECTION OF INTERACTIVE SYSTEMS: METHODS AND EXPERIENCES

Chairperson: Paul Oliver
Department of the Navy
Washington, DC

INCORPORATING REMOTE TERMINAL EMULATION INTO THE FEDERAL ADP PROCUREMENT PROCESS
Tom Wyrick
FEDSIM/NA
Washington, DC

Gerald Findley
General Services Administration
Washington, DC

Tuesday, October 24 (continued)

CPE TECHNIQUES APPLIED TO THE SELECTION OF A TIME-SHARING COMPUTER SYSTEM
Marshall Abrams
National Bureau of Standards
Washington, DC

H. Philip Hayden
Naval Ship R&D Center
Bethesda, MD

BENCHMARKING IN SELECTION OF TIME-SHARING SYSTEMS
D. J. M. Davies
University of Western Ontario
London, Ontario

OVERVIEW OF PROBLEMS IN REMOTE TERMINAL EMULATION
Vijay Trehan
Digital Equipment Corporation
Maynard, MA

Wednesday, October 25

PREDICTION PART I: METHODS 1978

Chairperson: Sam Fuller
Digital Equipment Corporation
Maynard, MA

A FORMAL TECHNIQUE FOR ANALYZING THE PERFORMANCE OF CONCURRENT SYSTEMS
John Sanguinetti
Digital Equipment Corporation
Maynard, MA

A METHOD FOR EVALUATING DIGITAL SYSTEM DESIGN ALTERNATIVES
Taylor L. Booth
University of Connecticut
Storrs, CT

PERFORMANCE EVALUATION OF A DIGITAL SYSTEM USING A PETRI NET-LIKE APPROACH
Y. W. Han
Bell Telephone Labs.
Naperville, IL

CONTROL THEORETIC APPROACH TO COMPUTER SYSTEM PERFORMANCE IMPROVEMENT
R. K. Jain
Digital Equipment Corporation
Maynard, MA

Wednesday, October 25 (continued)

PREDICTION PART II: QUEUEING-BASED

AN INVESTIGATION OF SEVERAL MATHEMATICAL
MODELS OF QUEUEING SYSTEMS
Rollins Turner
Digital Equipment Corporation
Maynard, MA

ON THE BUSY PERIOD OF A QUEUEING NETWORK OF TWO SERVICE
STAGES WITH EXPONENTIALLY DISTRIBUTED SERVICE TIME
R. K. Ma, and G. J. Stroebel
IBM General Systems Division
Rochester, MN

PREDICTION PART III: APPLICATIONS

Chairperson: A. Agrawala
University of Maryland
College Park, MD

MARKOVIAN MODEL OF A JOB
Jeff Mohr, and A. Agrawala
University of Maryland
College Park, MD

FORECASTING COMPUTER UTILIZATION
H. Pat Artis
Bell Telephone Labs.
Piscataway, NJ

CASE STUDY IN CAPACITY PLANNING - ANALYSIS OF AN
AUTOMATED BIBLIOGRAPHIC RETRIEVAL SYSTEM
R. P. Goldberg, A. I. Levy, and H. S. Schwerk, Jr.
BGS Systems, Inc.
Lincoln, MA

Thursday, October 26

PERFORMANCE IMPROVEMENT PART I: QUANTITATIVE METHODS

Chairperson: A. K. Jain
Bell Telephone Labs.
Holmdel, NJ

MULTIDIMENSIONAL DATA ANALYSIS AS A TOOL FOR THE
STUDY OF COMPUTER SYSTEMS
Anne Schroeder
IRIA LABORIA
France

AN APPLICATION OF TIME-SERIES ANALYSIS IN COMPUTER
PERFORMANCE EVALUATION
Major R. W. Kulp, and Major Kenneth Melendez
Air Force Institute of Technology
Dayton, OH

Thursday, October 26 (continued)

ESTIMATION OF RUN TIMES USING SIGNATURE
TABLE ANALYSIS
S. A. Mamrak, and P. K. Amer
Ohio State University
Columbus, OH

SENSITIVITY ANALYSIS AND THE RESPONSE
SURFACE OF A SIMULATION MODEL
Major Kenneth Melendez, and Major A. H. Linder
Air Force Institute of Technology
Dayton, OH

A STATISTICAL APPROACH TO RESOURCE CONTROL
IN A TIME-SHARING SYSTEM
C. A. MacKinder
University of Edinburgh
Edinburgh, Scotland

PERFORMANCE IMPROVEMENT PART II: APPLIED STATISTICS

Chairperson: Colonel R. A. Lejk
Air Force
Wright-Patterson AFB, OH

A STATISTICAL COMPARISON OF THE PERFORMANCE
EFFECTS OF TWO CONFIGURATIONS
Madhav Marathe
Digital Equipment Corporation
Maynard, MA

RELIABILITY MODELING OF COMPUTER SYSTEMS
Lloyd R. Hasche, and Richard A. Grace
Offutt AFB, NB

ANALYSIS OF VARIABILITY IN SYSTEM ACCOUNTING DATA
D. J. M. Davies
University of Western Ontario
London, Ontario

PERFORMANCE IMPROVEMENT PART III: MEASUREMENT APPLICATIONS

Chairperson: Phil Howard
Applied Computer Research
Phoenix, AZ

A RELATIVE ENTROPY-LIKE MEASURE FOR
SYSTEM ORGANIZATION
Jamie Chaikin, and R. A. Orchard
Bell Telephone Labs.
Piscataway, NJ

Thursday, October 26 (continued)

PRELIMINARY MEASUREMENT OF C.MMP UNDER A SYNTHETIC LOAD
P. F. McGehearty, and George Rolf
Carnegie-Mellon University
Pittsburgh, PA

Sam Fuller
Digital Equipment Corporation
Maynard, MA

PERFORMANCE STUDY OF A MINICOMPUTER SYSTEM
S. K. Lee, R. M. Maguire, and L. R. Symes
University of Regina
Regina, Canada

TO MP OR NOT TO MP
M. Lieberman
Chase Manhattan Bank
New York, NY

Friday, October 27

PANEL: IMPACT OF PRESIDENTIAL REORGANIZATION PROJECT RECOMMENDATIONS ON THE ADP LIFE CYCLE

Chairperson: Roxann Williams
Department of Agriculture
Washington, DC

Representatives of Government and Industry will discuss recommendations of the Presidential Reorganization Project and the impact they may nave on the federal ADP community. The Director of the Project, Mr. Walter W. Haase of the Office of Management and Budget, will be the introductory speaker. He will be followed by panel members discussing the potential effect of the recommendations on ADP operations, management, and procurement.

CONFERENCE WRAP-UP

CPEUG 78 PROGRAM
TRACK B

TUTORIALS

Tutorial Coordinator: John Bennett
Digital Equipment Corporation
Maynard, MA

Tuesday, October 24

A REVIEW OF WORKLOAD CHARACTERIZATION
A. Agrawala
University of Maryland
College Park, MD

COMPUTER SYSTEM SELECTION
S. A. Mamrak, and R. D. Amer
Ohio State University
Columbus, OH

Wednesday, October 25

USE OF MODELS FOR CAPACITY PLANNING
Jeffrey Buzen
BGS Systems, Inc.
Lincoln, MA

CAPACITY PLANNING
H. Pat Artis
Bell Telephone Labs.
Piscataway, NJ

IMPLEMENTATION OF THE CAPACITY PLANNING PROCESS
LeeRoy Bronner
IBM Systems Capacity Planning Department
Gaithersburg, MD

Thursday, October 26

COMPUTER PERFORMANCE MANAGEMENT

Chairperson: Philip J. Kiviat
SEI Computer Services
Washington, DC

DEVELOPMENT OF A TUNING GUIDE
Barry Wallack
Command and Control Technical Center
Washington, DC

GUIDANCE FOR SIZING ADP SYSTEMS
Mitch Spiegel, Dennis Gilbert, and James Mulford
FEDSIM
Washington, DC

Thursday, October 26 (continued)

HUMAN PERFORMANCE EVALUATION IN THE USE OF FEDERAL COMPUTER SYSTEMS: RECOMMENDATIONS
Mark Underwood
Navy Personnel R&D Center
San Diego, CA

TUTORIALS

THE BASICS OF COMPUTER PERFORMANCE MANAGEMENT
Captain J. C. Toole
Air Force
Gunter AFS, AL

A DATA CENTER PRODUCTION CONTROL SYSTEM
J. H. Garrett, Jr.
Value Computing
Merchantville, NJ

USING SAS AS A UNIFIED LANGUAGE FOR RETRIEVAL MANAGEMENT AND STATISTICAL ANALYSIS OF COMPUTER PERFORMANCE DATA
Bill Gjiertsen
SAS Institute
Raleigh, NC

PROBLEMS IN REMOTE TERMINAL EMULATION
Vijay Trehan
Digital Equipment Corporation
Maynard, MA

Friday, October 27

WORKSHOP: PERFORMANCE STATE-OF-THE-ART AND THE ADP LIFE CYCLE

Chairperson: Dennis Conti
National Bureau of Standards
Washington, DC

This workshop will begin with an introduction to the ADP life cycle. Attendees will break into subgroups with predefined objectives to address. Subgroups will later regroup and discuss results. This workshop provides the attendees with both an opportunity to influence next year's conference and to identify unsolved problems on which performance research and development should be focusing.

NBS-114A (REV. 11-77)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET**	1. PUBLICATION OR REPORT NO. NBS SP-500-41	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE COMPUTER SCIENCE & TECHNOLOGY Computer Performance Evaluation Users Group CPEUG 14th Meeting	5. Publication Date October 1978
	6. Performing Organization Code
7. AUTHOR(S) Editor: James E. Weatherbee	8. Performing Organ. Report No.
9. PERFORMING ORGANIZATION NAME AND ADDRESS NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234	10. Project/Task/Work Unit No.
	11. Contract/Grant No.
12. Sponsoring Organization Name and Complete Address *(Street, City, State, ZIP)* Same as No. 9.	13. Type of Report & Period Covered Final
	14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

The Proceedings record the papers that were presented at the Fourteenth Meeting of the Computer Performance Evaluation Users Group (CPEUG) held October 24-27, 1978 in Boston. The technical presentations were organized around the three phases of the ADP Life Cycle: the Requirements Phase (workload definition), the Acquisition Phase (computer system and service selection), and the Operational Phase (performance) measurement and prediction methods). The program of CPEUG 78 is also included and serves as an Applendix to the Proceedings.

17. KEY WORDS *(six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons)* ADP life cycle; computer performance evaluation; computer performance measurement; computer performance prediction; computer system acquisition; conference proceedings; CPEUG; hardware monitoring; on-line system evaluation; prediction methods; queuing models; simulation; software monitoring; workload definitio

18. AVAILABILITY [X] Unlimited	19. SECURITY CLASS (THIS REPORT) UNCLASSIFIED	21. NO. OF PAGES 353
[ ] For Official Distribution. Do Not Release to NTIS		
[X] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Stock No. SN003-003	20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED	22. Price $6.00
[ ] Order From National Technical Information Service (NTIS) Springfield, Virginia 22151		

USCOMM-DC 66035-P78

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, D. C. 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name ______________________________

Company ______________________________

Address ______________________________

City ______________ State ______________ Zip Code ______________

(Notification key N-503)