DEPARTMENT OF COMMERCE ★ NATIONAL BUREAU OF STANDARDS ★

NBS SPECIAL PUBLICATION 401

U.S. DEPARTMENT OF COMMERCE / National Bureau of Standards

Computer Performance Evaluation

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau consists of the Institute for Basic Standards, the Institute for Materials Research, the Institute for Applied Technology, the Institute for Computer Sciences and Technology, and the Office for Information Programs.

THE INSTITUTE FOR BASIC STANDARDS provides the central basis within the United States of a complete and consistent system of physical measurement; coordinates that system with measurement systems of other nations; and furnishes essential services leading to accurate and uniform physical measurements throughout the Nation's scientific community, industry, and commerce. The Institute consists of a Center for Radiation Research, an Office of Measurement Services and the following divisions:

Applied Mathematics — Electricity — Mechanics — Heat — Optical Physics — Nuclear Sciences[2] — Applied Radiation[2] — Quantum Electronics[3] — Electromagnetics[3] — Time and Frequency[3] — Laboratory Astrophysics[3] — Cryogenics[3].

THE INSTITUTE FOR MATERIALS RESEARCH conducts materials research leading to improved methods of measurement, standards, and data on the properties of well-characterized materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; and develops, produces, and distributes standard reference materials. The Institute consists of the Office of Standard Reference Materials and the following divisions:

Analytical Chemistry — Polymers — Metallurgy — Inorganic Materials — Reactor Radiation — Physical Chemistry.

THE INSTITUTE FOR APPLIED TECHNOLOGY provides technical services to promote the use of available technology and to facilitate technological innovation in industry and Government; cooperates with public and private organizations leading to the development of technological standards (including mandatory safety standards), codes and methods of test; and provides technical advice and services to Government agencies upon request. The Institute consists of a Center for Building Technology and the following divisions and offices:

Engineering and Product Standards — Weights and Measures — Invention and Innovation — Product Evaluation Technology — Electronic Technology — Technical Analysis — Measurement Engineering — Structures, Materials, and Life Safety[4] — Building Environment[4] — Technical Evaluation and Application[4] — Fire Technology.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides technical services designed to aid Government agencies in improving cost effectiveness in the conduct of their programs through the selection, acquisition, and effective utilization of automatic data processing equipment; and serves as the principal focus within the executive branch for the development of Federal standards for automatic data processing equipment, techniques, and computer languages. The Institute consists of the following divisions:

Computer Services — Systems and Software — Computer Systems Engineering — Information Technology.

THE OFFICE FOR INFORMATION PROGRAMS promotes optimum dissemination and accessibility of scientific information generated within NBS and other agencies of the Federal Government; promotes the development of the National Standard Reference Data System and a system of information analysis centers dealing with the broader aspects of the National Measurement System; provides appropriate services to ensure that the NBS staff has optimum accessibility to the scientific information of the world. The Office consists of the following organizational units:

Office of Standard Reference Data — Office of Information Activities — Office of Technical Publications — Library — Office of International Relations.

---

[1] Headquarters and Laboratories at Gaithersburg, Maryland, unless otherwise noted; mailing address Washington, D.C. 20234.
[2] Part of the Center for Radiation Research.
[3] Located at Boulder, Colorado 80302.
[4] Part of the Center for Building Technology.

Computer Performance Evaluation

Proceedings of the Eighth Meeting of
Computer Performance Evaluation
Users Group [CPEUG]

Sponsored by
United States Army Computer
Systems Command

and

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

Edited by

Dr. Harold Joseph Highland

State University Agricultural and
Technical College at Farmingdale
New York 11735

U.S. DEPARTMENT OF COMMERCE, Frederick B. Dent, *Secretary*

NATIONAL BUREAU OF STANDARDS, Richard W. Roberts, *Director*

Issued September 1974

Library of Congress Cataloging in Publication Data

Computer Performance Evaluation Users Group.
Computer performance evaluation.

(NBS Special Publication 401)
"CODEN: XNBSAV."
Sponsored by the U. S. Army Computer Systems Command and the Institute for Computer Sciences and Technology, National Bureau of Standards.
Supt. of Docs. No.: C 13.10:401
1. Electronic digital computers—Evaluation—Congresses. I. Highland, Harold Joseph, ed. II. United States. Army. Computer Systems Command. III. United States. Institute for Computer Sciences and Technology. IV. Title. V. Series: United States. National Bureau of Standards. Special Publication 401.

QC100.U57 No. 401 [QA76.5] 389'.08s
[001.6'4]
74-13113

National Bureau of Standards Special Publication 401

Nat. Bur. Stand. (U.S.), Spec. Publ. 401, 155 pages (Sept. 1974)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1974

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
(Order by SD Catalog No. C13.10:401). Price $1.80

FOREWORD

As part of the Federal Information Processing Standards Program, the Institute for Computer Sciences and Technology of the National Bureau of Standards sponsors the Federal Information Processing Standards Coordinating and Advisory Committee (FIPSCAC) and a series of Federal Information Processing Standards (FIPS) Task Groups. The Task Groups are composed of representatives of Federal departments and agencies, and in some cases of representatives from the private sector as well. Their purpose is to advise NBS on specific subjects relating to Information Processing Standards.

One of the Task Groups is FIPS Task Group 10 - Computer Component and Systems Performance Evaluation. Among FIPS Task Group 10's other responsibilities is to sponsor a self-governing Computer Performance Evaluation User's Group (CPEUG) whose purpose is to disseminate improved techniques in performance evaluation through liaison among vendors and Federal ADPE users, to provide a forum for performance evaluation experiences and proposed applications, and to encourage improvements and standardization in the tools and techniques of computer performance evaluation.

With this volume, the proceedings of the CPEUG are for the first time being made available in a form that is readily accessible not only to Federal agencies but to the general public as well. This is in recognition of the fact that computer performance evaluation is important not only for Federal ADP installations but also for those of the private sector.

It is expected, therefore, that this volume will be useful in improving ADP management through the use of performance evaluation in both the private and public sectors.

Acknowledgement is made of the assistance of Dr. Joseph O. Harrison, Jr., Chairman of the FIPSCAC, and Captain Michael F. Morris, USAF, Chairman of FIPS Task Group 10, in promoting and producing this volume.

Ruth M. Davis, Ph.D., Director
Institute for Computer Sciences
and Technology
National Bureau of Standards
U. S. Department of Commerce

ABSTRACT

The Eighth Meeting of the Computer Performance Evaluation Users Group (CPEUG), sponsored by the United States Army Computer Systems Command and the National Bureau of Standards, was held December 4-7, 1973, at NBS, Gaithersburg. The program chairman for this meeting was Merton J. Batchelder of the U.S. Army Computer Systems Command at Fort Belvoir, Virginia 22060 (CSCS-ATA Stop H-14).

About 150 attendees at this meeting heard the 17 papers presented on computer performance, evaluation and measurement. Among the papers presented were those dealing with hardware and software monitors, workload definition and benchmarking, a report of FIPS Task Force 13, computer scheduling and evaluation in time-sharing as well as MVT environment, human factors in performance analysis, dollar effectiveness in evaluation, simulation techniques in hardware allocation, a FEDSIM status report as well as other related topics.

These proceedings represent a major source in the limited literature on computer performance, evaluation and measurement.

Key words: Computer evaluation; computer performance; computer scheduling; hardware monitors; simulation of computer systems; software monitors; systems design and evaluation; time-sharing systems evaluation

PREFACE

The evolution and rapid growth of computer performance, evaluation and measurement has been the result of an amalgam of developments in the computer field, namely:

- the growing complexity of modern digital computer systems;
- the refinement of both multiprogramming and multiprocessing capabilities of an increasing number of computer systems;
- the concomitant expansion and complexity of systems programming;
- the accelerated trend toward the establishment of computer networks;
- the myriad of applications programs in business, technical, scientific, industrial and military fields; and
- the emphasis on massive data bases for information retrieval, data analysis, and administrative management.

Prior to proceeding, it is essential to define several basic terms so that the examination is not made in a vacuous environment.

Performance, according to the dictionary, is defined as "an adherence to fulfill, to give a rendition of, to carry out an action or a pattern of behavior."

Evaluation is defined as a "determination or the fixing of a value, to estimate or appraise, to state an approximate cost."

Measurement is defined as a "process of measuring, the fixing of suitable bounds, the regulation by a standard."

Computer performance, evaluation and measurement is now vital to the designer, the user and the management-owner of a modern computer system. To some, computer performance, evaluation and measurement is a tool, a marriage of abstract thought and logic combined with the techniques of statistical and quantitative methods. To others, it is a technique with very heavy reliance on modeling and simulation and simultaneously involves features of both classical experimentation and formal analysis. The problem of exact specification is made the more difficult by the recent birth and development of computer performance, evaluation and measurement as a discipline within computer science.

Among the early practitioners of this discipline were the members of **CPEUG** [Computer Performance Evaluation Users Group], individuals from many United States Governmental agencies involved in various phases of this field. At about the same time, there were a number of academicians as well as analysts from business and industry working in this area, and this gave rise to the formation within the Association for Computing Machinery of **SIGME** [Special Interest Group in Measurement and Evaluation] which is currently known as **SIGMETRICS.**

In its formative period of growth, computer performance, evaluation and measurement has relied heavily upon modeling and simulation. It has been concerned with the design of computer systems as well as the analysis of the behavior of these systems. The analyst, on the one hand, has used simulation to secure detailed information about a system that he has created, or about which his knowledge is limited. On the other hand, the analyst has used simulation to test various hypotheses about the system in an effort to improve its performance. It is a quixotic hope that as this discipline grows it will become more interdisciplinary and involve the work not only of modelers, simulators and statisticians, but also behavioral scientists, economic analysts and management scientists.

Computer performance, evaluation and measurement is replete with benefits for the computer community – the manufacturer of modern electronic processing equipment, the user of that equipment, especially the manager responsible for the complete operations, as well as the purchasers of this equipment. It is capable of providing many urgently needed answers to problems faced today, such as:

- procurement – the evaluation of various proposals for different systems configurations;
- planning – determination of the effects of projected future workloads upon an existing or planned system;
- costing – detailed information of cost data to users of an existing or planned computer system or network;
- scheduling – the allocation of systems resources to the various demands on the system;
- designing – how best to design a new or modify an existing system, and how to improve the operation of that system;
- optimization – determination of a systems mode to enhance the performance of a portion or of the entire system.

Within this volume are the technical papers presented at the **Eighth Meeting of the Computer Performance Evaluation Users Group.** The Meeting was sponsored by both the United States Army Computer Systems Command and The Institute for Computer Sciences and Technology of the National Bureau of Standards, and was held December 4th – 7th, 1973 at Gaithersburg, Maryland. Added to these technical papers are other papers which were presented at an earlier meeting of CPEUG but never published.

Dr. Harold Joseph Highland, Editor

Chairman, **ACM\ SIGSIM** [Special Interest Group in Modeling and Simulation of the Association for Computing Machinery]

Chairman, Data Processing Department of the State University Agricultural and Technical College at Farmingdale ⟨ New York ⟩

June 1974

TABLE OF CONTENTS

* PREFACE
Dr. Harold Joseph Highland, Editor
New York State University College at Farmingdale v

* COMMENTS OF CHAIRMAN, CPEUG
John A. Blue
Department of Navy, ADPESO x

* COMMENTS OF PROGRAM CHAIRMAN
Merton J. Batchelder
U. S. Army Computer Systems Command xii

* KEYNOTE ADDRESS
Brigadier General R. L. Harris
Management Information Systems, U. S. Army 1

* INITIATING COMPUTER PERFORMANCE EVALUATION

GETTING STARTED IN COMPUTER PERFORMANCE EVALUATION
Philip J. Kiviat and Michael F. Morris
Federal Computer Performance Evaluation and Simulation Center 5

A METHODOLOGY FOR PERFORMANCE MEASUREMENT
Daniel M. Venese
The MITRE Corporation 15

* ACCOUNTING PACKAGES

USE OF SMF DATA FOR PERFORMANCE ANALYSIS AND RESOURCE ACCOUNTING ON IBM LARGE-SCALE COMPUTERS
Robert E. Betz
Boeing Computer Services, Inc. 23

USING SMF AND TFLOW FOR PERFORMANCE ENHANCEMENT
James M. Graves
U. S. Army Management Systems Support Agency 33

* SOFTWARE MONITORS

USACSC SOFTWARE COMPUTER SYSTEM PERFORMANCE MONITOR: SHERLOC
Philip Balcom and Gary Cranson
U. S. Army Computer Systems Command 37

* BENCHMARKS

BENCHMARK EVALUATION OF OPERATING SYSTEM SOFTWARE: EXPERIMENTS ON IBM'S VS/2 SYSTEM
Bruce A. Ketchledge
Bell Telephone Laboratories 45

REPORT ON FIPS TASK GROUP 13 WORKLOAD DEFINITION AND BENCHMARKING
David W. Lambert
The MITRE Corporation 49

* HARDWARE MONITORS

PERFORMANCE MEASUREMENT AT USACSC
Richard Castle
U. S. Army Computer Systems Command 55

A COMPUTER DESIGN FOR MEASUREMENT -- THE MONITOR REGISTER CONCEPT
Donald R. Deese
Federal Computer Performance Evaluation & Simulation Center 63

* SIMULATION

THE USE OF SIMULATION IN THE SOLUTION OF HARDWARE ALLOCATION PROBLEMS
W. Andrew Hesser
USMC Headquarters, U. S. Marine Corps 73

* RELEVANT CONSIDERATIONS

HUMAN FACTORS IN COMPUTER PERFORMANCE ANALYSES
A. C. (Toni) Shetler
The Rand Corporation 81

DOLLAR EFFECTIVENESS EVALUATION OF COMPUTING SYSTEMS
Leo J. Cohen
Performance Development Corporation 85

* SPECIFIC EXPERIENCE

COMPUTER SCHEDULING IN AN MVT ENVIRONMENT
Daniel A. Verbois
U. S. Army Materiel Command 99

PERFORMANCE EVALUATION OF TIME SHARING SYSTEMS
T. W. Potter
Bell Telephone Laboratories 107

A CASE STUDY IN MONITORING THE CDC 6700 - A MULTI-PROGRAMMING, MULTI-PROCESSING, MULTI-MODE SYSTEM
Dennis M. Conti
Naval Weapons Laboratory 115

* FEDERAL COMPUTER PERFORMANCE EVALUATION AND SIMULATION CENTER

FEDSIM STATUS REPORT
Michael F. Morris and Philip J. Kiviat
Federal Computer Performance Evaluation & Simulation Center 119

* PREVIOUS MEETING: SPECIAL PAPERS

DATA ANALYSIS TECHNIQUES APPLIED TO PERFORMANCE MEASUREMENT DATA
G. P. Learmonth
Naval Postgraduate School 123

A SIMULATION MODEL OF AN AUTODIN AUTOMATIC SWITCHING CENTER COMMUNICATIONS DATA PROCESSOR
LCDR Robert B. McManus
Naval Postgraduate School 127

* ATTENDEES: EIGHTH MEETING OF COMPUTER PERFORMANCE EVALUATION USERS GROUP 137

COMMENTS OF CHAIRMAN

COMPUTER PERFORMANCE EVALUATION USERS GROUP

This publication provides papers presented at the eighth meeting of the Computer Performance Evaluation Group held at the National Bureau of Standards, sponsored by the United States Army Computer Systems Command and the Institute for Computer Sciences and Technology of the National Bureau of Standards; and some papers presented at the meeting held in March 1973 at Monterey, California, and sponsored by the Naval Postgraduate School.

As the now past chairman of CPEUG, I am very pleased at the qualities of these papers. They provide valuable information on a wide spectrum of computer performance evaluation techniques and tools. The authors deserve a great deal of credit for the time and effort they have expended on the preparation and presentation of their papers. I thank them very much for making the meetings that I have chaired so successful. I know that with that kind of support, future meetings will be even more interesting and informative. I look forward to having continued contact with all members of CPEUG, and I wish the new officers all the best.

John A. Blue

COMMENTS OF PROGRAM CHAIRMAN

EIGHTH MEETING OF

COMPUTER PERFORMANCE EVALUATION USERS GROUP

I wish to express my appreciation to the members of government, industry and the academic community who contributed their time and resources to make this a highly successful conference.

Brigadier General Richard L. Harris, Director of Management Information Systems for the Army, set the tone of the conference with his keynote address. Authors of the 16 technical papers shared the benefits and problems of their endeavors with Computer Performance Evaluation tools and techniques.

The conference technical program began with an overview of items for consideration when initializing a computer performance evaluation program. This was followed by applications of the CPE tool s in order of increasing complexity. Philip Kiviat, Technical Director of the Federal Computer Performance Evaluation and Simulation Center, concluded with a summary of contributions by FEDSIM.

The conference scheduling of a presentation each hour allowed time for questions and participation from the audience. Ample coffee breaks also permitted time to pursue specific CPE problems and discussions and stimulate new ideas.

This exchange of performance evaluation experiences should have widespread effect on our ADP community. Each agency is striving to improve its application of ADP systems. Interchange of experience should continue through contacts with others met at the conference.

Good luck, and let us hear from you.

Mert Batchelder

COMPUTER PERFORMANCE EVALUATION USERS GROUP

EIGHTH MEETING HELD 4-7 DECEMBER 1973

AT NATIONAL BUREAU OF STANDARDS

GAITHERSBURG, MARYLAND

KEYNOTE ADDRESS

Brigadier General R. L. Harris

Director, Management Information Systems, US Army

Good Morning Ladies and Gentlemen:

I am pleased to have a chance to welcome you to this meeting of the Computer Performance Evaluation Users Group. I believe such meetings are most beneficial to all users of ADP through the sharing of information on mutual problems and because they provide us the chance to leave our desks and, for a little while, to look at the bigger problems on the horizon. Since becoming the Army's Director of Management Information Systems in July, I have had an opportunity to review some of the history and current philosophy which underlies the Army's approach to the management of ADP. I have also discussed with my staff the future trends and areas for exploration over the next five to ten years. I would like to share some of these thoughts with you this morning.

The Army has had a long association with the development and use of digital computers. Initially we used these machines to assist us in processing complex scientific computations and, more recently, our attention has been focused on automating the bread and butter functions---logistics, personnel administration, and financial management. The Army's first interest in digital computers dates back to 1942 when we commissioned the University of Pennsylvania's Moore School of Electrical Engineering to design and produce the electronic numerical integrator and computer (the ENIAC). The installation of the ENIAC in 1947 in the Ballistic Research Labs at Aberdeen, Maryland marked the beginning of the widespread use of electronic computing machines within the Federal Government and the civilian economy as well. The ENIAC was closely followed by the installation of our second computer, the EDVAC, or Electronic Discrete Variable Automatic Computer, operational at BRL since 1949. These initial computers were used primarily for solution of ballistic equations, fire control problems, data reduction, and related scientific problems. From these beginnings, the Army expanded its use of both unique and off-the-shelf computers to other scientific and administrative areas, so that by the beginning of the sixties, the Army had over 100 computers in its inventory.

During the sixties, the Army expanded its use of ADP at a rate of almost fifteen percent annually. This expansion was caused by two major factors--the conflict in Southeast Asia and the increasing complexity of our weaponry and the personnel skills and logistic base necessary to support them. This complexity was the price we had to pay for the greater move, shoot, and communicate equipment of the Army in the field. More importantly, the attendant requirement for faster, more responsive ways of supporting this combat power became quickly apparent.

We have, however, been conscious of the need to economize and have made significant gains in our efforts to stem this spiral of ever-increasing ADP costs. Like most of you, we have been helped along by Congress, of course. However, in spite of these efforts, our resource commitment is still significant. We are currently expending more than three hundred and eighty million dollars and nineteen thousand manyears on this effort, while operating almost 950 computers of all shapes and sizes. This commitment to automation has been so pervasive that there is probably no manager at any level within the Army whose decisions are not influenced by the products of our automated systems.

To continue to provide the qualitative support required by today's managers in an era of decreasing resource availability and increasing constraints on flexibility, we all must become better at managing the resources we do have. Our increased management efforts in the Army will fall in three major areas: better planning, better management of systems development, and, lastly, increased productivity in software.

From an organization viewpoint, we shall continue our commitment to the centralization of software development through increased emphasis on the development of standard systems in more functional areas. The benefits which the Army has gained since the establishment of its central design agencies have permitted us to reduce the total number of personnel in ADP by almost one-third while at the same time increasing the qualitative support to managers at our major operating activities. Combined with our centralization efforts is our continuing search for better methods of defining functional requirements. Part of this effort has been to place squarely in the hands of the functional user the responsibility for determining these requirements and for preparation of the supporting economic justification for undertaking the software development effort.

Better software development is predicated upon better planning methods which can help us to allocate our planned resources to meet new user requirements. Along these lines, we believe that we can no longer sell automation by using subjective arguments. We all recognize that the day of automation for automation's sake has passed. We must now support our requirements with quantitative economic and benefit analyses, as well as workload projections. Use of both of these tools is predicated upon an adequate means of data gathering for predictive purposes. In line with this, one of the current major efforts of my office is to improve our dollar and utilization reporting systems to provide the data we now feel is necessary to help us make these critical resource allocation decisions. One such effort which we have just completed is a hardware requirements projection model called SCORE. This model enables us to project needed upgrades in computer capability dependent upon increasing workload over time.

Turning now to the question of software productivity, it is this area in which we can most benefit from the discussions which will take place over the next four days. Increased productivity will not come from increased or better management alone. Technology will assist us and we in the Army intend to lean on the technologists. In order to do this, however, we will have to increase our management of R&D expenditures in information science. One of the investments we are making is in the area of performance monitors. We are excited about the possibilities inherent in performance monitors, and have been since we first started using them in 1970. We are currently decreasing our operating costs through the use of performance monitors at our central design agencies. Details of these efforts are included in two presentations by the Computer Systems Command. We also have two monitors at our Computer Systems Support and Evaluation Command (our ADPE evaluation and selection command). CSSEC is using these monitors in an ombudsman role to help determine the right hardware configuration to match user processing requirements. We are using the monitors in a number of ways -- no different, I'm sure, than how they are being used by others. We have found the use of these monitors most effective as aids in ADPE configuration balancing and the optimization of data base design.

We are particularly interested in their potential in software optimization. The trend in software development, including systems software, is toward potential use of a higher order language. (UNIVAC, for example, for their new, but unannounced, 1100 system are building most of their compilers and operating system in PL-1). In this environment, monitors are a prerequisite in achieving throughput effectiveness; that is, finding the critical code and optimizing it. We are attempting to build our software in a similar manner. We are still subjected to the argument that software effectiveness can only be achieved through use of an assembly language. I believe that management, ever sensitive to total life cycle costs, will find that the higher level language approach, in conjunction with an effective monitor, will prove the more economic solution for most systems.

I certainly cannot be classified as a computer expert, but it is clear to me that the automation field in this next decade will undergo considerable change. There will certainly be greater emphasis on the computer-communications interrelationship and continued moves toward distributive processing with hierarchies of processing power. Revolutionary changes are on the horizon in software. We all recognize the accelerating cost imbalance between hardware and software. Steps must be taken to redress this imbalance. Software development must move from its present cottage industry stage if it is to continue to exist. The engineers, armed with microprogramming techniques and a well established manufacturing discipline, are a force to be reckoned with. The general purpose off-the-shelf computer as we know it may very well give way in the eighties to microprogrammed special purpose devices capable of solving well defined problems at minimal cost.

What does the software industry need to meet this challenge? From my perspective, it appears to me that the list includes:

a. A simple management-oriented language for describing automation requirements--a tool which can be placed in the hands of the computer neophyte to enable him to bridge the functional analyst-computer analyst communication gap.

b. Means of increasing the productivity of our programmers. Many attempts have been undertaken to achieve this goal. Monitors used in conjunction with a procedure-oriented language will help. The team programmer concept, structured programming, the machine tool approach to software development and automatic programming are all attempts which have been undertaken to increase programming productivity.

c. Programming is estimated to represent only fifteen percent of the system life cycle time. Approximately fifty percent of a programmer's time is spent trying to find logical (and illogical) errors. Very little attention is now being focused on the validation problem, and yet, it is precisely here where our major resources are expended. This subject area demands our professional attention.

d. Improved control systems for management use are required. A prerequisite to management control is adequate planning. Planning, in turn, requires refined estimating techniques. The computer industry has been embarrassingly deficient in its failure to develop a system to help management estimate system costs. The DOD community has made minimal attempts in this area with little success to show for our money. This area, too, demands much more of our managerial attention.

The challenges I have outlined are substantial, but the potential return is great. We have no alternative but to approach them -- if necessary -- one at a time.

This symposium is certainly a step in that direction, and I urge all of you to keep these objectives in mind as you participate in these discussions.

Thank you.

GETTING STARTED IN COMPUTER PERFORMANCE EVALUATION

Philip J. Kiviat and Michael F. Morris

FEDSIM

1. WHEN TO START A PROGRAM

A. EXPENDITURE LEVEL DECISION

When the cost of a computer installation reaches some threshold value, a CPE program may be examined as an investment problem. The installation costs include all computer and peripheral equipment; supplies such as cards, tapes and paper; all computer operations personnel; and any systems programmers that are used solely to keep the computer system operating. The value of the system's products must be established. Where no price is already associated with existing products, a break-even price per customer-ordered piece of output must be calculated and assumed as the start-up value. Once a system's cost exceeds its value, or value could be higher if the system could produce more salable products, CPE is needed. When value exceeds cost, but cost is high, CPE aimed at cost reduction is in order. As long as no value is associated with the system's products, there will be no real cost-reduction motivation to attempt to improve a system's performance (except, perhaps, as an academic exercise).

B. PROBLEM DRIVEN DECISION

More typical of the reason most CPE efforts are begun: another set of applications are to be added to an already-busy computer system; projected workload growth was an underestimate; the director of operations says he needs another or a larger computer; etc. Someone must look deeply and carefully into the way that existing resources are being applied to meet existing demands. Then the demands and resources must be aligned so that room for growth is available. A simple example may be seen that really made a substantial difference in work performed at no real change in products produced: A system of 14 batch programs that had been operational on various equipment for nearly 16 years was modelled using a simulation package. This system ran an average of 4 hours and 30 minutes and interfaced with several other systems so that neither inputs nor outputs could be changed. Several proposed changes were tested on the simulator. Two very simple changes were found that made substantial differences in the system's performance: first, the programs were allowed to run simultaneously (multiprogram) wherever possible; and, second, records stored on disk were blocked to the average track size. The average system run time went down to 2 hours 5 minutes giving better than a 50% reduction each time the system is run. Total CPE project time was about six manweeks; five in the simulation phase and one to alter and test the programs. No magic or profound intellect was involved in this effort, just a thorough examination of the demands and resources.

C. PRELIMINARY ANALYSES: SETTING GOALS, DEFINING APPLICATIONS

One of the worst ways to start a CPE effort is to purchase a hardware monitor and assign a person to use it to "do good things". Many starts have been made just like this and few, if any, have succeeded. The monitors are stored away, collecting only dust. It is much more reasonable if specific systems are identified for analysis with modest goals set such as -- decrease the time that is used by one old program by 5%; or, document the flow of jobs form the input desk to the output window and reduce the total turnaround time for one large job by 5%. No sophisticated tools are needed and any standard time study text will provide guidelines for the test. What these types of projects allow a manager to accomplish are, first, operations personnel become aware of a systematic improvement effort; and, second, recommendations will be made regarding information that is needed, but very difficult to obtain. Setting only one or two simple improvement goals causes unknowns to become visible. These unknowns may then be categorized into application areas for solution by specific CPE tools or techniques. The usual result of these initial efforts is establishment of a systematic analysis of the accounting package information and some physical rearrangement of computer support room layout. Once the analysis effort is underway, specific needs for additional CPE capability will become apparent. This approach is really quite conservative, and it may take six months or more before the effort begins to produce useful results. But, once underway, very valuable results will be produced.

2. SELECTING TEAM MEMBERS

The work "team" is used continuously to stress the point that a lone individual seldom succeeds for very long in a CPE effort. This is not to imply that more than one full-time person is always necessary, but rather that at least three different types of individuals need to be involved and the background that leads to CPE successes is seldom found in any one person.

A. NECESSARY BACKGROUND

Implementing improvements is often more difficult than identifying improvements. Nearly all significant improvements require either programming skill or a background in systems engineering. Since systems programmers usually evolve from applications programmers, system men are ideal CPE specialists. A systems programmer with a field engineering history is even better. Education in a scientific discipline usually helps a person to examine situations systematically. A scientific education nearly always exposes an individual to fundamentals of mathematics and statistics. These traits--systematic thinking and knowledge of math and statistics--are also necessary for a CPE team. An additional feature that adds immeasurably to a CPE team's effectiveness is an academically diversified background. Reasoning here is like in any other developing field--the broader the investigator's background, the more likely that parallels will be seen in other fields where similar problems have already been solved.

B. TYPES OF PEOPLE

Considerable amounts of nearly original or, at least, innovative thought is a major part of any disciplined investigation. (And the CPE team will indeed appear to be investigators if they are properly placed in the organizational structure.) Picking a person and making him think as a routine part of a task is a nearly impossible approach. It is far better to pick a reaearch type who takes every assignment as a challenge to his thought process. (If no researchers are available, pick a lazy person with the experience and education mentioned earlier.)

Next, and at times most important, is a person who can explain and convince. Perhaps an "evangelist" would come closer to describing this person than "salesman", but "evangelist" would be a misleading outline heading. (That is, no divine guidance is necessary for CPE success--it does help, though.) This person is particularly important at the beginning and end of each CPE project. The idea of performing each project usually needs to be "sold" all the way down to the lowest involved working level. In fact, to be of lasting worth, it is more important that these working level people be convinced that CPE is worthwhile than that higher management be so convinced. Further, a sales type is essential when a project is completed (to make certain that those responsible for authorizing implementation of team recommendations are thoroughly convinced to do so). Unless changes are made, the CPE effort will be wasted.

As a CPE team must produce useful work in a timely manner, a worker is a must. This is the one individual who should be full time in the CPE field. He need not be particularly well educated or intelligent but he must not be lazy. He will probably be both the easiest one to find and the least expensive to pay. This is the team member who should have the strong system programming background as he will be the one that will sift through reams of output and determine feasible changes in programs.

It should by now be clear that good CPE teams aren't easy to find. This is true. In two cases of large CPE activities, about 100 detailed resumes of people who think they are qualified are examined to find each CPE specialist who turns out to be good. Most often, it's faster to offer the duty to someone in your shop that has the background and is of the type described earlier (or maybe to steal someone who is already a success - but this expensive).

C. IMPORTANCE OF ATTITUDE

It is often true that very good systems programmers are extremely reluctant to change anything that they didn't think of first (or, at least to change it the way someone else has). This is a very poor attitude to bring into a CPE team. The best team members are open-minded and eager to examne suggested changes from whatever source. Be careful that an individual who is too conservative in his work doesn't defeat the purpose of your CPE team. It is an imaginative and innovative field.

D. IN-HOUSE OR CONSULTANTS

If time is available to decide to

go into CPE based on return on investment (as opposed to an immediate operational problem) it is certainly more beneficial to develop the CPE capability in-house. If the CPE need is immediate and if you have no capability in-house and can't hire a proven performer as a regular employee, then the only alternative is to use a consultant. There are certainly cases where a return-on-investment decision requires that the CPE investment be less than the cost of an established in-house effort. Such cases could be handled by consultants (although these aren't the jobs consultants try very hard to get...). As a general rule, solve immediate one-time problems with consultants. Solve immediate, but general problems with a consulting contract that requires the consultant to train your in-house CPE team. Solve continuing, non-urgent problems totally in-house. The suggestion here is no different for a CPE team than it would be for any other effort -- the best way is in-house.

3. ORGANIZATIONAL PLACEMENT OF THE CPE TEAM

Let's assume here that computer system expenditures are large enough to justify a continuing 1 1/2 - 2 manyear CPE investment per year on an in-house basis. (In the consultant case, the following could be put in terms of "Who should the consultant report to?").

A. LINE VERSUS STAFF

A CPE team performs a staff function: it investigates existing methods, finds and recommends alternative methods, and outlines ways of implementing methods. Although the team must be able to implement any recommendation, it is usually a misallocation of scarce talent for the team to do this. Usually, and particularly in very large data processing environments, the CPE team will periodically be challenged (generally by a middle-manager in applications programming surroundings) to do the implementation job themselves. About every 8 to 12 months, this should be done. In the process of implementing its own recommendations, the team not only gains operational credibility, it also has the opportunity to sharpen and update the team's own programming (or systems design) skills. This keeps the CPE team from becoming a group of <u>former</u> experts. And this is important.

B. REPORTING AUTHORITY

Organizations are so very different that the only way of describing placement, in general, is in relative terms. The team should, at the very lowest, report to the same person that the data processing or computer center manager reports to. Ideally, the CPE team should report to the corporate-level controller. This placement insures that recommendations that are accepted will be implemented. Also, it will encourage the team to view their activity in a cost-benefit light. The more cost conscious the CPE team, the more likely that sound recommendations will be made. Unless the data processing center is extremely large, the CPE team should not report to the manager of data processing. (Extremely large: more than three separate and different computer systems <u>and</u> monthly least cost--or equivalent--of more than $100,000.) If there is a good argument for using consultants continuously rather than in-house personnel, it is in this reporting authority region: A CPE team is really a working level activity. It is not customary to have this type activity reporting at such a high level. However, it is customary for consultants and outside auditors to circulate at the working level and report at very high management levels. Except in cases of very small companies (the president acting as data processing manager), company size has little to do with the CPE team's organizational placement. It is more a matter of scaling the CPE effort to match the computer effort, which is generally dependent on company size.

C. CHARTER

This is the CPE team's license to operate. When duties to identify improvements in computer system operations are assigned to a group outside the direct control of the computer center manager (as recommended earlier), it is mandatory that clear notice of the team's operational arena be given. This notice should be formal (written) and signed by the highest possible corporate officer. Such a notice will be a charter or license to become involved in even the most routine aspects of providing computer support. The charter need not be long or particularly specific. General direction is usually better because it's never known ahead of time exactly which areas will produce the most plentiful savings. The ideas regarding system bottlenecks and workload characteristics that grow in the computer center without benefit of CPE tools, are almost always wrong. (Explain: scientific vs. business mix, printer-bound, etc.) A charter should include, as a minimum, the following statements:

"No new programs or systems of programs and no substantial changes to existing programs are to be implemented without first having the design or change reviewed by the CPE team. If in the CPE team's opinion it is necessary, the design or change must be simulated to predict its impact on the existing workload and to test any reasonable design alternatives before committing resources to programming and implementing the system or change.

"Before ordering new equipment to replace, to add to, or to enhance existing equipment's speed or capacity, the CPE team must be called upon to measure the levels of activity and contentions of the system or portions of the system that would be affected by the new equipment."

This type of charter will allow the team to examine both existing systems and proposed changes before new workloads or equipment acquisitions are committed. Determining the impact of such changes requires a thorough knowledge of existing workloads and systems. The benefit from this initial learning phase for the CPE team will be significant: it is almost always the first time that a review of all facets of the computer center's activity will have been attempted by knowledgeable and impartial persons. This phase is, in itself, sound justification for establishing a CPE team.

D. CONTROLS

Since a good CPE team that is properly placed in the organization will represent a high level of technical competence and have the endorsement of upper management, there will be a tendency to react quickly to the team's suggestions. It is, therefore, important that rather strict controls be established to insure that the team follows a total project plan (more about this later) to its conclusion before discussing possible recommendations with those who will be implementing the changes. Otherwise, there may be changes well underway before all aspects of the changes are examined. And these may turn out to be unacceptable changes when everything is considered. It is quite easy to avoid this situation by adding a statement to the "charter" that establishes a need for written directive by some specified manager (above the CPE team) before any recommendation is to be implemented. Whenever possible, management should insist that the CPE team provide at least two recommended approaches to the solution of any problem. And the approach should be selected by a manager above the CPE team or within the area that will be most affected by the recommended change.

4. PROJECT ADMINISTRATION

A. JOB DESCRIPTIONS

The formal writeups that cover CPE team members must mention the need to (1) work with documentation describing the logical and physical flow of signals through the system and its components; (2) perform measurement analyses and coding changes of applications and control programs; (3) install or connect such CPE devices or tools as are available; (4) develop and use simulation and modeling techniques; and (5) document activities and recommendations for use by management. These may be elaborated as required to fit each personnel department's peculiar needs. But these five basic areas are necessary to insure that the CPE team has the minimal capability to perform useful projects.

B. RESPONSIBILITIES

A CPE team just as any other group of employees, must be expected to produce results that are worth more than they cost. A detailed listing of responsibilities and procedures will be given in a moment, in classical management terms. At this point, it is enough to say that the team is responsible for "paying its own way." In the early start-up phase, it isn't reasonable to demand this of the team. But, by the end of its first year of existence, a CPE team that hasn't documented savings equal to or greater than its costs should be dissolved. By itself, this is a heavy responsibility to meet.

C. PROCEDURES

The day-to-day operating procedures of the CPE team are essentially the same as for an audit group or, in times past, for a time-study effort: define a problem, examine existing methods, postulate changes, test change, recommend best changes, and oversee implementation of accepted recommendations. Written procedures are important especially for groups that visit detached sites.

D. RECOMMENDING CHANGES

As pointed out earlier, changes should be recommended to management, not at the working level. The recommendations should only reach the working level in clear-cut, directive form.

E. FORMAL REPORTS

These will be the only real (tangible) product of a properly run CPE team. The changes (except in the rare cases mentioned already) will be made by other programmers and engineers. Only by carefully and completely documenting each project can the CPE team progress in an orderly manner. A format for project reports that has proven useful in several environments is that of the classical "staff study": Purpose, Scope, Procedure, Findings, Conclusions, and Recommendations -- preceded by a short summary of major findings and recommendations. Hopefully, the entire report, but at least the summary and recommendations, should be completely "jargon" free. (The best report is worthless if only the authors can understand it.)

F. PROJECT MANAGEMENT

It has been implied several times that the CPE team is most effective if it operates on a project basis. CPE projects tend to be ad hoc in nature: each one has a single purpose. The following remarks are therefore framed in this project orientation. In classical management terms the CPE project manager's job is to:

--PLAN: Determine requirements of the project in terms of people, facilities, equipment, etc.; estimate the project's duration and relationships between work-load of the CPE team and the project's deliverables.

--ORGANIZE: Request and insure availability of the necessary facilities, equipment, personnel, material, and (most important) authority.

--DIRECT: Set time and cost milestones and bounds for what must be done; make all operating, project-specific decisions.

--CONTROL: Measure performance against plan and take any necessary actions to correct malperformance.

--COORDINATE: Insure that all involved activities are aware of, and receptive to the project's efforts, goals, and recommendations.

To make certain that the full impact of this project administration section has been made clear, a few minutes will be spent on an exhaustive listing of the responsibilities of a CPE project manager. First, it is pointed out that if the CPE team is large, or if team members have specific skills that not all team members have, each project or study may have a different project manager. Each project manager is responsible for the technical content and conduct of the study. His ultimate responsibility is twofold: to the managers that receive the team's recommendations and to the individual or group who will implement the team's recommendations. For lack of a better term, these two groups are referred to as the "customer". The individual CPE project manager is responsible for:

1. Insuring that he can detect malperformance.

2. Establishing measures to prevent malperformance.

3. Assuring that he can correct malperformance.

4. Insuring that all ideas will be explored and exploited.

5. Exercising positive cost control.

6. Insuring that schedules will be met.

7. Establishing quality control procedures.

8. Thoroughly understanding the kinds of management abilities that will be required in a project.

9. Determining the quantity of management required in a project.

10. Determining what must be provided by all parties involved, e.g., sutomer, other suppliers.

11. Knowing what is needed to "solve" the problem.

12. Understanding the customer's problem and translating it into a solvable CPE problem.

13. Insuring that a clear, consistent, and appropriate plan for each study is produced.

14. Knowing how the results of the study will fill the customer's needs.

15. Insuring that the planned approach to the project is logical and realizable.

16. Understanding the details of the approach.

17. Insuring that all essential tasks are included.

18. Insuring that no unnecessary tasks are included.

19. Knowing the output from each task.

20. Judging whether or not each task is the best way to achieve the output.

21. Knowing what resources are required for each task.

22. Periodically reviewing the adequacy of the skills, quantity of personnel, facilities, equipment, and information.

23. Insuring that commitments of all resources made to the project are honored.

This is a rather long list for each CPE project but the team's tasks are complex and detailed enumeration of the individual project manager's responsibilities is important. What, then, is left for the leader of the CPE team to do if he is not always the project leader? -- Even a longer list of non-technical, management tasks.

The CPE team leader acts to:

1. Manage objectives of each CPE study.

   -Does the customer know what he wants?

   -Does the customer have the authority to perform its tasks?

   -Are all the results the customer requests needed?

   -Is the customer adequately motivated to support CPE efforts over the long haul?

2. Schedule the project tasks.

   -Use a documented, quantitative (graphic) method to schedule people, equipment, facilities, tasks.

   -Act to acquire resources when needed.

   -Establish procedures to monitor and control progress.

3. Select and organize personnel.

   -Determine when outside support is required.

   -When necessary, establish a project management structure.

   -Motivate personnel to the goals of the project.

   -Distribute workload to make the best use of present talents and promote cross training by on-the-job experience.

4. Manage the scope of each study.

   -Determine whether a study should be specific to a project or made more general.

   -Determine essential components and boundaries of the study.

   -Insure all boundary effect assumptions are considered and stated explicitly in project reports.

5. Control the level of detail for each study.

   -Derive an evolutionary development plan that permits continual assessment and redirection.

   -Strive for a minimal, consistent, level of detail, adding detail only on evidence that it is required.

6. Control costs.

   -Establish control procedures.

   -Guard against inefficient use of people and equipment.

   -Obtain CPE resources best suited to a project at a cost that is consistent with project size and importance.

7. Manage validation phase.

   -Expose all assumptions.

   -Determine the "level of noise" inherent in different methods.

   -Establish procedures for debugging and testing.

   -Specify nominal outputs in advance of experimentation.

   -Insure sensitivity analyses are performed.

   -Specify levels of confidence required to demonstrate success of a method.

   -Insist a validation plan be developed in advance of its need.

8. Manage documentation effort.

   -Insure that all assumptions and limitations are specified.

   -State the purpose and use of each recommendation.

   -Describe each recommendation clearly in narrative form.

   -Establish a project notebook that is a chronological log of all project decisions, actions, trip reports, etc.

   -Insure that a good channel of communication exists between CPE team members and their customers.

9. Manage experimentation effort.

   -Develop an experiment plan that is consistent with project objectives before starting a new type of project.

-Establish validity of statistical analysis procedures when they are applied.

-Insure analyses are consistent and correct.

-Insure summaries and presentations are clear, illuminating and concise.

-Insure experiments produce required results.

5. ACQUIRING TOOLS AND TECHNIQUES

(Most of the rest of the outlined topics will be covered in detail by other presentations. They are mentioned here, briefly, simply to introduce them as important topics that require concern if a viable CPE effort is to be established at any installation).

A. FREE TOOLS - ACCOUNTING PACKAGES

Before any special CPE tools are purchased or techniques are learned, the output of the system accounting package should be studied in detail and thoroughly understood. A tremendous amount of information that can lead to improved computer performance is contained in accounting data. In almost every case, it is a "free" good. That is, it is collected as a routine task. If nothing is now done to your facility's capacity: eliminate the accounting package. But if you are seriously considering the establishment of a CPE activity, you'll have to reinstall this package at least periodically. Nevertheless, if you don't use its output now, stop generating it.

B. COMMERCIALLY AVAILABLE TOOLS

A list of all CPE suppliers is given in the following table. There are three general product categories:

-Accounting package reduction programs, referred to in the table as "Other".

-Monitors, broken into hardware and software monitors.

-Simulation, which is in the form of general purpose CPE packages, specialized simulation languages, and simulation studies.

C. SURVEYING THE FIELD

Several individuals have compared and contrasted the various CPE tools that are now available. Such surveys should be examined before commitments are made to any one CPE supplier. User's groups are the best sources of this survey information. Figure 8-10 lists several CPE product suppliers as an indication of the broad availability of such products.

D. LEARNING TECHNIQUES

All CPE techniques require some training before they can be used. In order of least to most training, these are:

-Accounting system analysis

-Software monitors

-Hardware monitors

-Simulation languages

-Simulation packages

While CPE techniques can be learned by self-study, classroom training is advised. On-the-job training is imperative, as classroom courses can teach only <u>technique</u>, and not guidelines or standards for its application in the field.

E. PROCURING TOOLS

As mentioned earlier, it is not wise to begin a CPE effort by purchasing CPE equipment. First, use the available accounting data, then acquire a software monitor or accounting data reduction package (lease one, don't buy, as it will soon be unused in most shops), then develop a simulation capability, and finally -- when the CPE team justifies it for a specific project -- acquire a hardware monitor. This procurement of tools should span at least one year and probably two or three.

6. A FLOW CHART FOR CPE DECISIONS

The chart shown in Figure 8-17 describes many situations where CPE may be useful. It's certainly not exhaustive, but it covers enough situations to give a new CPE team an ambitious start. Its use can help an organization to decide where the application of CPE may produce worthwhile results.

CONTINUING THE EVALUATION ACTIVITY

Once established, the CPE team loops through a series of continuing activities:

A. Examine and select new CPE tools

1. To keep up with developments.
2. To interchange information with CPE teams

B. Determine CPE measurement frequencies with respect to:

1. The characteristics of each installation

   a. Capital investment
   b. Workload stability
   c. Personnel

2. The aging of systems

3. Planned workload changes

   a. New systems

   b. New users

   c. New user mix, e.g., RJE, interactive, batch

C. Maintain a close watch over the distribution of computer activity by device and by program to:

1. Relate equipment costs to level of use

2. Determine trade-offs possible between cost and performance

3. "Tune" the total system as a routine task

   a. Data set allocation

   b. Selection of programs for multi-programming

   c. Operating system and applications program optimization

D. Develop a CPE data base

1. CPE information collected by one tool will usually be used in conjunction with that collected by another tool.

2. Historical data can be used to predict trends in workload, usage, etc.

3. As new analysis tools and techniques are developed, data from well-understood situations is needed to test their discriminant and predictive power.

4. A common data format and structure is needed to compare similar installations for normative and management purposes.

5. Customized interfaces can be built between the data base and the CPE tools used to standardize and simplify tool usage and add additional features, e.g., editing.

6. "Trigger" reports can be incorporated into data base processing or interface programs to indicate that something is happening in the system that should be studied in detail.

E. Integrate operations and measurement activities

1. Educate operations personnel to be aware of visual cues to system malperformance

2. Use existing and new operations reports

   a. Console logs

   b. System Incidence Reports

   c. "Suggestion Box"

3. Educate operators in use of tools to improve selection of jobs for multiprogramming, mounting of tapes and disk packs upon request, etc.

Operator performance is an overriding factor in the operation of an efficient third-generation computer installation.

F. Establish good relations with the computer system vendor

1. May require assistance for probe point development

2. May require permission to attach a hardware monitor

3. First instinct of maintenance personnel is to resist measurement activities

4. Acquire a knowledge of his activities and the acquaintance of his research staff

G. Kiviat Graphs for Diagnosing Computer Performance (Refer to reprint of "Computerworld" article)

SPECIAL CONSIDERATIONS AND FINANCIAL ASPECTS

A. A minimum investment in CPE tools and technicians will be one manyear per year and an equivalent in tool and training expenditures, computer costs, travel expenses, etc. Assume this is $40,000 based on proration of the following costs experienced at the Federal Computer Performance Evaluation and Simulation Center (FEDSIM).

1.	Typical FEDSIM staff salary:	$19,700
2.	Software monitor:	$1-15,000
3.	Hardware monitor:	$4-100,000
4.	Accounting system analysis program:	$3-15,000

5. Simulation packages:
$12-40,000

B. CPE savings must be at least 5%, probably no more than 20$ on an annualized basis, although many individual projects will show larger returns. For example:

$40,000/.05 = $800,000 per year - to justify on low expected CPE value;

$40,000/.20 = $200,000 per year - to justify on high expected CPE value.

C. CPE savings are difficult to quantify in many instances, e.g., system design analysis via simulators, configurations studies for system enhancement, measurement of workload to prepare benchmarks for new RFP, etc.

CPE savings are easy to quantify in other instances, e.g., equipment returned after a reconfiguration prompted by a measurement analysis, computer time saved by system and program tuning, etc.

But what is extra computer capacity, improved turnaround or better response worth to the company?

D. Manufacturers will not assist most smaller installations in their CPE efforts, as their goal is to increase these to larger systems. CPE is counter to this goal.

E. There are many small systems using general purpose computers for highly specialized applications. CPE effort in these installations should be spent in stripping all unused or unnecessary parts out of the control program to match it to the special application. In the case of serial, batch systems (or serial, batch equipment) there is little that CPE can do for performance. In these cases, the best available programming techniques are called for.

F. CPE should include:

1. Software package evaluation

2. Software validation and verification

3. Equipment reliability studies

4. Studies of programming techniques

5. Management and organizational effects

But it rarely does. CPE should be viewed from a <u>total systems perspective</u>.

A METHODOLOGY FOR PERFORMANCE MEASUREMENT

Daniel M. Venese

The MITRE Corporation

I. INTRODUCTION

This paper presents a methodology for performance evaluation. A general framework is developed to orient and direct a performance evaluation task. Within this framework, operating systems can be classified and compared, hypotheses formulated, experiments performed, and cost/benefit criteria applied. This methodology has the following underlying assumptions:

(a) Most systems can be modified or tuned to achieve improved performance.

(b) Although some optimizations will not degrade other areas of system performance (removing a system-wide bottleneck), most performance gains will be the result of compromise and cause degradation in another area of system performance.

(c) Relatively small changes can often lead to dramatic changes in system performance.

(d) Modifications indicated by a performance evaluation study should be cost effective.

(e) In many cases when a performance bottleneck is removed, another will immediately become the limiting factor on performance.

The aim of this methodology is to account for the significant factors that affect a performance evaluation effort. Inadequate performance evaluations result from limited outlooks. Purely technical evaluations concentrate on hardware and software monitors, but ignore the human factors. Other efforts have obtained huge quantities of data without a framework for analysis. In general, two of the most common inadequacies are to take a narrow approach to performance evaluation and to gather data without having a means to analyze it.

II. OUTLINE OF METHODOLOGY

The following outline describes the steps in the performance evaluation methodology:

(a) Understand the system. Classify the subject operating system according to the framework provided in paragraph III.

(b) Survey the environment. Examine and analyze the environment surrounding the computer installation including management attitudes and goals, mode of operation, workload characteristics, and personnel experience.

(c) Evaluate system performance:

(1) examine problem types,

(2) formulate performance evaluation hypothesis,

(3) conduct benefits analysis, and

(4) test performance evaluation hypothesis.

III. OPERATING SYSTEMS FRAMEWORK

In order to conduct an effective performance evaluation, it is necessary to understand the operating system. Although the precise implementation of each operating system differs in many respects, they have many facets in common. The identification and classification of these common functions into a framework for understanding operating systems is an important step in computer performance evaluation (CPE).

This framework for operating systems is not intended to be all inclusive since operating systems can be examined and classified from other meaningful points of view. Examining and classifying from the user point of view, is an example of another framework. The framework presented here isolates those portions that are important in CPE and allows various systems to be compared by means of this common classification.

Large-scale third-generation computers typically have more resources than any single program is likely to need. The rationale for an operating system is to manage the utilization of these resources allowing many programs to be executing at one time. The resource manager concept of operating system is the product of J. Donovan and S. Madnick of MIT.

In a computing system with finite resources and a demand for resources that periodically exceeds capacity, the resource manager (operating system) makes many policy decisions. Policy decisions are strategies for selecting a course of action from a number of alternatives. There is general agreement that as many relevant factors as possible should be included in policy decisions. For example, the core allocator algorithm might consider such factors as the amount requested, the amount available, the job priority, the estimated run time, other outstanding requests, and the availability of other requested peripherals. Disagreement arises as to how factors should be weighted and the strategies that are most appropriate for the installation workload.

The component of the operating system that decides which jobs will be allowed to compete for the CPU is the job scheduler. The scheduling policy might be first-in, first-out (FIFO) or estimated run time with the smallest jobs scheduled first. The FIFO strategy is one of the simplest to design and implement, but it has many disadvantages. The order of arrival is not necessarily the order in which jobs should be selected for execution nor can it be assured that a FIFO scheduling policy will provide a balanced job mix and adequate turnaround for priority jobs. Although general prupose algorithms can provide an acceptable level of service for a wide spectrum of situations, tuning to the needs of the particular DPI can realize significant gains in efficiency and ease of use.

Once a job has been allocated and is eligible for execution, the process scheduler (system dispatcher) decides when jobs gain control of the CPU and how long they maintain control. The process scheduler queue may be maintained in order by priority, by the ratio of I/O to CPU time or in a number of different ways. The job scheduler decides which jobs will be eligible to compete for CPU time while the process scheduler decides which jobs will receive CPU time. A number of other decision points and important queues are found in operating systems. These include I/O supervisors, output queues, interrupt processing supervisors, and data management supervisors.

Another important facet of operating systems is the non-functional software which carries out the policy decisions and comprises the bulk of the operating system software. The gains to be effected in the areas of policy making are primarily the result of tuning to the particular Data Processing Installation (DPI) while the gains from the non-functional software are primarily the result of improvements in efficiency. For example, the routine that physically allocates core has a number of minor housekeeping decisions to make and any improvements to be realized in this area will be from an improved chaining procedure or a faster way of searching tables or a similar type of gain in efficiency.

Once this framework for computer systems has been adopted, systems are no longer viewed as a collection of disparate components, but as a collection of resources. The operating system is the manager of these resources and attempts to allocate them in an efficient manner through policy decisions and non-functional software to carry out these decisions. System bottlenecks are not mysterious problems, but are the result of excess demand for a limited resource. By stating performance problems in terms of the relevant resources instead of in terms of the effects of the problem, the areas in need of investigation are clearly delineated.

IV. SURVEY THE ENVIRONMENT

Understanding and classifying the subject operating system is one facet of understanding the total system. Computers do not operate in a vacuum. Their performance is influenced by the attitudes and abilities of the operations personnel, programmers and managers. Although improving the performance of a system by streamlining the administrative procedures is not as dramatic as discovering a previously unknown inefficiency in the operating system, the net gain can be just as worthwhile.

Computer Operations

Parts of the following survey are based on the Rand document, "Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort," by T. E. Bell.

(a) Determine the prime operational goal of the computer center.

(b) Determine which job or group of jobs constitutes a significant part of the installation's workload.

(c) Describe the explicit actions taken by the computer operations to help schedule the computer, e.g., using job priorities to promote a good job mix.

(d) Determine the most frequent causes for computer reruns, e.g., mounting the wrong tape for a production run.

(e) Determine from interviewing computer operations personnel the extent to which the following are done:

  (1) scheduling of jobs,

  (2) premounting of tapes and disks,

  (3) interfacing with users,

  (4) card handling,

  (5) responding to console commands, and

  (6) documenting operational difficulties.

(f) Determine the extent and impact of on-line applications on the computer system.

(g) Determine from interviewing users the adequacy of the following: turnaround on batch jobs, response time for time-sharing and on-line applications, and the availability of the system.

(h) Determine the average number of system failures per day, the average time before the system is available and the average time between system failures.

(i) Determine a typical working schedule for the computer center and include preventative maintenance, dedicated block time, availability of time-sharing and on-line applications, and time devoted to classified jobs.

(j) Determine the approximate lines of print output per day, number of cards punched, and number of tapes processed.

(k) State how priority users are scheduled.

(l) State the number of special forms required each day.

Computer System

(a) Determine the approximate CPU utilization on a daily basis.

(b) Determine the average number of jobs processed per day.

(c) Determine the average number of jobs being multiprogrammed/multiprocessed.

(d) Construct a diagram of the computer configuration.

(e) Determine the degree of use of on-line (disk) storage as compared to tape for the following:

(1) temporary (scratch) files,

(2) object programs,

(3) source programs,

(4) short-term storage,

(5) permanent storage, and

(6) scratch storage for sorts.

Job Characteristics

(a) Determine the typical peripheral requirements for the median computer job.

(b) Determine the typical core and CPU requirements for the median computer job.

(c) Determine the percentage of jobs that are production and debug.

(d) Determine the percentage of jobs that are classified or that require special processing.

(e) Determine the percentage of jobs that use a language processor.

(f) Determine the percentage of jobs that are batch, time-sharing, and on-line.

(g) Determine and examine in detail the program or programs that require(s) the most significant percentage of computer resources.

(h) Determine the language and percentage of use for the applications programs.

Measurement Activities

(a) State which performance evaluation tools are used to evaluate system performance.

(b) Determine the current or last performance evaluation project.

(c) State how hardware and software problems are recorded, monitored, and resolved.

(d) State what system modifications have been made to improve performance.

(e) Determine what reports are available on a periodic or as required basis on system performance.

(f) Determine what performance evaluation tools are available at the installation.

V. PROBLEM TYPES

This section describes the various kinds of problem areas encountered in a CPE effort. The classification of problems as operational, administrative, hardware or software is essentially for the purpose of discussion. Although problems can impact in many areas, corrective action should be taken at the cause instead of treating the effects.

Operations

The area of computer operations is one of the most important to CPE. In reality, computer operations can have a dramatic effect on system performance. The typical net gain of an improvement in software may be 4% to 8% while the actions of computer operations in the areas of scheduling, job control, peripheral operation, etc., can affect performance by 10% to 20%.

There are several measures historically used to gauge the effectiveness and productivity of computer installations. These indicators of performance include the number of jobs processed per day, the average CPU utilization, the average turnaround time, and the system availability. Although these measures apply with varying degrees to other areas, they seem most appropriate for computer operations since they are most affected by computer operations. Since examples can be found of systems delivering overall poor performance that have high CPU utilization or that process many jobs a day, these indicators should be considered in the context of the total installation.

In general, well run computer operations will have the following characteristics in common:

(a) The number of informal or undocumented procedures is at a minimum.

(b) The existing procedures are well publicized to the users, are easy to use, and are kept current.

(c) Whether the DPI is "open" or "closed" shop, access to the computer room is based primarily on actual need.

(d) Procedures are established to manage and monitor the computer configuration. This might include down time and error rate per device, utilization per device, and projections of system changes based on workload.

(e) The service to users of the DPI meets their needs.

(f) The areas of responsibility for the operators, supervisors, and job control personnel are well defined.

Administrative

Administrative problems are non-technical factors that degrade system performance. Included under this heading are the attitudes and goals of management: the organizational structure of the

DPI, the training and education of the data processing staff, and the level of documentation for computer systems. Initial analysis performed on data gathered by monitoring tools will not indicate that administrative problems are limiting performance, but further analysis supplemented by the experience of the analyst will usually narrow the potential list of causes to a few and through the process of elimination administrative problems can be isolated.

In a system of any kind and especially in computer systems, each of the components should work harmoniously with others. To some degree, the operator can compensate for inadequate documentation and the job scheduler can provide a reasonable job mix even when not tuned to the DPI workload, but beyond that point the effectiveness and performance of the total system will be degraded. When the operators spend a disproportionate amount of time trying to decipher documentation, other areas will be neglected; and when the job scheduler makes assumptions about the DPI workload that are incorrect, the total throughput will be reduced.

Software

The types of software problems that can occur are of two kinds, system software problems as in operating systems or access methods and application software problems or those that are associated with a particular set of user developed programs. System problems relating to performance are potentially more critical since they can affect all the programs run on the system while application program problems usually do not have a system-wide impact. If, however, the application program is a real-time application that runs 24 hours a day, then its performance assumes a high degree of importance.

System Software

Third generation computers have general purpose operating systems that can accommodate a wide spectrum of workloads with varying degrees of efficiency. A problem is that inefficiencies in large-scale computers are often transparent to the user. A heightened awareness of tuning techniques can lead to improved performance.

Performance problems associated with system software include:

(a) The suitability of the scheduling algorithm to the DPI workload.

(b) The selection of resident and transient system routines.

(c) The interface of the system to the operator via the computer console.

(d) The tape creation, processing, and label checking routines.

(e) System recovery and restart procedures.

(f) Direct access file management techniques, e.g., file creation, pruging access.

(g) The operation of data management routines.

(h) The suitability of the access routines, e.g., indexed sequential, random to the DPI workload.

Applications Software

While the tuning of system software has a system-wide benefit, the tuning of applications programs has benefit for that particular application. At first, it seems that the investigation of performance problems for application programs would be an area of low payoff since tuning the entire system has an inherently greater impact than tuning a subset of the system. The amount of payoff is dependent upon the size of the application program and the amount of resources used. Tuning a system dedicated to a single application is virtually equivalent to tuning the system software. The characteristics of the DPI workload will dictate the selection of application programs for tuning and performance evaluation. Care should should be taken not to lose sight of the areas of high payoff while concentrating on areas of intrinsically limited payoff. An example is a system that was tuned in regard to record and block sizes, file placements on disk and still delivered poor performance. What was overlooked was the fact that the system contained 12 sorts. Reducing the number of sorts would have equaled all the other performance gains so painstakingly achieved.

The following list itemizes tuning techniques for application programs:

(a) Tuning programs to the special features and optimization capabilities of language processors.

(b) Attempting to balance the ratio of I/O to CPU.

(c) Adjusting file volumes, record lengths, and block sizes to the peculiarities of the physical media.

(d) Ordering libraries so that the most frequently used members are searched first.

(e) For virtual paged environments the following guidelines are applicable:

  (1) Locating data and the instructions that will use it in close proximity to maximize the chance of being on the same page.

  (2) Structuring the program logic so that the flow of control is linear and that references are concentrated in the same locus.

  (3) Minimizing page breakage by avoiding constants, data, and instructions that overlap pages.

  (4) Isolating exception and infrequently used routines on separate pages from the mainline program logic.

(5) Attempt to construct programs around a working set of pages that will most likely be maintained in core by the page replacement algorithm.

(f) Determine the feasibility of allocating scratch, sort, and intermediate files on permanently mounted media.

(g) Key indicators that may indicate inefficiently written application programs are the following:

(1) Unusually high CPU utilization may indicate unnecessary mathematic operations and excessive data conversions.

(2) Imbalanced channel utilization may indicate poor file dispersal.

(3) High seek time may indicate poorly organized files.

Hardware

Indicators that hardware problems exist include: poor overall reliability, failure of key components, frequent backlogs of work, imbalanced channel utilization, low channel overlap, and I/O contention for specific devices. These indicators can be grouped into three general problem types:

(a) misuse of configuration which implies that the basic hardware is adequate, but that it is improperly structured so that some channels sit idle while others are saturated or that the availability of key components is low because of poor configuration planning,

(b) inadequate configuration which implies that the hardware components lack the capacity, speed, or capability to provide a minimum level of service and that it is advisable to procure an expanded or revised configuration, and

(c) excess capacity which implies that DPI workload is being processed inefficiently.

Hardware problems are corrected by upgrading, eliminating or reconfiguring specific components. Alternatives exist to hardware changes such as reducing or smoothing the workload, tuning the system, adopting a new operating system, but the increasing cost of software versus the decreasing cost of hardware often dictates that the cost effective solution is a hardware change.

VI. FORMULATE HYPOTHESIS OF PROBLEM

Working hypotheses are guideposts to system evaluation. The purpose of the hypothesis is to provide a working solution to a specific problem as related to an understanding of the total system. The hypothesis directs the collection of data that will be used to affirm or deny its validity. Becuase of the complexity of computer systems, a number of hypothesis will be required to investigate interrelated system problems. Figure 1 is a chart which may be used to relate symptoms to causes and aid in the formulation of relevant hypothesis.

VII. BENEFITS ANALYSIS

This section provides guidelines to be used in determining the benefits to be realized from a performance improvement. In almost every case a performance improvement will come at some cost, whether it is the time and resources necessary to implement it, the degradation that will result in some other area of system performance or the change in mode of operation. Certain changes will result in such a dramatic gain in system performance that a detailed benefits analysis is not necessary to determine their value and other changes will produce such a marginal gain in performance that they are obviously not worth the effort. The difficult decisions are the borderline cases where a number of factors must be weighed before a judgement is reached.

The cost to implement a proposed change can be expressed in time and resources, degradation in some other area of system performance, or changes in operating and administrative procedures. Different types of costs can be associated with CPE changes. The operating system efficiency might be improved by recoding a frequently used system module in which case the cost would be the one-time expense to replace the module. A change in job log-in procedures might improve the ability to trace and account for jobs and this would be a recurring cost (since it would be done each time a job was submitted). In general, one-time or nonrecurring costs are usually effected in the areas of computer system hardware and software while recurring costs are usually effected in administrative and operating procedures.

VIII. TEST HYPOTHESIS

In a structured approach starting with an understanding of the total system, an analysis of the problem types, and a well formulated hypothesis, the amount and type of data needed to test the hypothesis should be limited and relatively easy to obtain. Data to test the hypothesis can be derived from personal observations and accounting data or explicit performance monitoring tools such as hardware and software monitors, simulations, analytical models, or synthetic programs.

The hypothesis is intended to be no more than a tentative working solution to the problem and should be reformulated as necessary when additional data is collected. To test a specific hypothesis data should be collected and experiments performed in a controlled environment.

Even when the data gathered tends to support the validity of the performance evaluation hypothesis, the net gain in performance may be surprisingly small or even negligible. The reason for this is that a second restraint on performance became the primary limiting factor once the most obvious was removed. The performance analyst should be aware that there are many limitations on performance and as one bottleneck is eliminated another will become evident. If, for example, the

SYMPTOMS	HIGH CPU	LOW CPU (WAIT)	LOW CPU/CHANNEL OVERLAP	LARGE SEEK	HIGH CHANNEL USE	LOW CHANNEL USE	LOW CHANNEL OVERLAP	CHANNEL IMBALANCE	LOW MULTIPROGRAMMING	INADEQUATE RESPONSE TIME
HIGH CPU	3		1, 2, 3							
LOW CPU (WAIT)		4		2						
LOW CPU/CHANNEL OVERLAP	1, 2, 3		9							
LARGE SEEK		2		5, 6, 10						
HIGH CHANNEL USE					5, 6, 9		8	8		
LOW CHANNEL USE						10	7	7		
LOW CHANNEL OVERLAP					8	7	2, 8			
CHANNEL IMBALANCE					8	7		2, 8		
LOW MULTIPROGRAMMING									1, 4, 11, 13	
INADEQUATE RESPONSE TIME										3, 9, 11, 12

CAUSES:

1. IMBALANCED SCHEDULING
2. POOR FILE PLACEMENT
3. INEFFICIENT CODING (APPLICATIONS)
4. INEFFICIENT OPS
5. EXCESSIVE PROGRAM LOADING
6. INEFFICIENT DATA BLOCKING
7. EXCESSIVE CHANNEL CAPACITY
8. POOR DEVICE/CHANNEL PLACEMENT
9. CPU BOUND WORKLOAD
10. I/O BOUND WORKLOAD
11. LIMITED CORE CAPACITY (INADEQUATE)
12. LIMITED CPU CAPACITY (INADEQUATE)
13. LIMITED I/O CAPACITY (INADEQUATE)

FIGURE 1

RELATING SYMPTOMS TO CAUSES

primary impediment to multiprogramming is the scheduling algorithm, then its redesign may achieve a significant gain in the multiprogramming factor, but only a small improvement in total performance because the addition of more jobs causes a key channel to become saturated and the I/O wait time to become excessive. A total approach will not stop with discovering the most obvious limitations to performance, but will delve deeply into the system and postulate a number of limitations on performance and approach them in a concerted manner.

BIBLIOGRAPHY

ACM-COMMUNICATIONS

1. A Method for Comparing the Internal Operating Speeds of Computers, E. Raichelson and G. Collins, (May 64, vol. 7, no. 5, pp. 309-310).

2. Relative Effects of Central Processor and Input-Output Speeds Upon Throughput on the Large Computer, Peter White, (December 64, vol. 7, no. 12, pp. 711-714).

3. Determining a Computing Center Environment, R. F. Rosin, (July 65, vol. 8, no. 7, pp. 463-468).

4. Economies of Scale and IBM System/360, M. B. Solomon, Jr., (June 66, vol. 9, no. 6. pp.435-440). Correction: (February 67, vol. 10, no. 2, p. 93).

5. Systems Performance Evaluation: Survey and Appraisal, Peter Calingaert, (January 67, vol. 10, no. 1, pp. 12-18). Correction: (April 67, vol. 10, no. 4, p. 224).

6. An Experimental Comparison of Time-Sharing and Batch Processing, M. Schatzoff, R. Tsao, and R. Wiig, (May 67, vol. 10, no. 5, pp. 261-265).

7. Further Analysis of a Computing Center Environment, E. S. Walter and V. L. Wallace, (May 67, vol. 10, no. 5, pp. 266-272).

8. An Experimental Model of System/360, Jesse H. Katz, (November 67, vol. 10, no. 11, pp. 694-702).

9. A Methodology for Calculating and Optimizing Real-Time System Performance, S. Stimler and K. A. Brons, (July 68, vol. 11, no. 7, pp. 509-516).

10. On Overcoming High-Priority Paralysis in Multiprogramming Systems: A Case History, David F. Stevens, (August 68, vol. 11, no. 8, pp. 539-541).

11. Some Criteria for Time-Sharing System Performance, Saul Stimler, (January 69, vol. 12, no. 1, pp. 47-53).

12. A Note on Storage Fragmentation and Program Segmentation, B. Randell, (July 69, vol. 12, no. 7, pp. 365-372).

13. The Instrumentation of Multics, Jerome H. Saltzer and John W. Gintell, (August 70, vol. 13, no. 8, pp. 495-500).

14. Comparative Analysis of Disk Scheduling Policies, T. Teorey, (March 72, pp. 177-184).

15. The Role of Computer System Models in Performance Evaluation, Stephen R. Kimbleton, (July 72, vol. 15, no. 7, pp. 586-490).

ACM-COMPUTING SURVEYS

16. Performance Evaluation and Monitoring, Henry C. Lucas, Jr., (September 71, pp. 79-91).

ACM-JOURNAL

17. Analysis of Drum I/O Queue Under Scheduled Operation in a Paged System, E. G. Coffman, (January 69, pp. 73-99).

ACM-PROCEEDINGS

18. Application Benchmarks: The Key to Meaningful Computer Evaluations, E. O. Joslin, (20th National Conference 65, pp. 27-37).

19. The Program Monitor - A Device for Program Performance Measurement, C. T. Apple, (20th National Conference 65, pp. 66-75).

20. Hardware Measurement Device for IBM System/360 Time-Sharing Evaluation, Franklin D. Schulman, (22nd National Conference 67, pp. 103-109).

21. A Hardware Instrumentation Approach to Evaluation of a Large Scale System, D. J. Roek and W. C. Emerson, (24th National Conference 69, pp. 351-367).

ACM-SIGOPS

22. Workshop on System Performance Evaluation, (April 71, Harvard University).

AFIPS-CONFERENCE PROCEEDINGS: SPARTAN BOOKS, NEW YORK

23. Cost-Value Technique for Evaluation of Computer System Proposals, E. O. Joslin (1964 SJCC, vol. 25, pp. 367-381).

24. System Aspect: System/360 Model 92, C. Conti, (1964 FJCC, vol. 26, pt. II, pp. 81-95).

COMPUTER DECISIONS

25. How to Find Bottlenecks in Computer Traffic, (April 70, pp. 44-48).

26. A Pragmatic Approach to Systems Measurement, Sewald, (July 71).

27. Use Measurement Engineering for Better Systems Performance, Philip G. Bookman, Barry A. Brotman, and Kurt L. Schmitt, (April 72, pp. 28-32).

COMPUTERS AND AUTOMATION

28. A Methodology for Computer Selection Studies, O. Williams, et. al., (May 63, vol. 12, no. 5, pp. 18-23).

29. Methods of Evaluating Computer Systems Performance, Norman Statland, (February 64, vol. 13, no. 2, pp. 18-23).

30. Computer Analysis and Thruput Evaluation, R. A. Arbuckle, (January 66, vol. 15, no. 1, pp. 12-15, 19).

31. Standardized Benchmark Problems Measure Computer Performance, John R. Hillegass, (January 66, vol. 15, no. 1, pp. 16-19).

MODERN DATA

32. The Systems Scene: Tuning for Performance, J. Wiener and Thomas DeMarco, (January 70, p. 54).

NATIONAL COMPUTER CONFERENCE

33. Performance Determination - The Selection of Tools, If Any, Thomas E. Bell, (1973, pp. 31-38).

34. A Method of Evaluating Mass Storage Effects on Systems Performance, M. A. Diethelm, (1973, pp. 69-74).

35. Use of the SPASM Software Monitor to Evaluate the Performance of the Burroughs B6700, Jack M. Schwartz and Donald S. Wyner, (1973, pp. 109-120).

36. A Structural Approach to Computer Performance Analysis, P. H. Hughes and G. Moe, (1973, pp. 109-120).

RAND REPORTS

37. Computer Systems Analysis Methodology: Studies in Measuring, Evaluating, and Simulating Computer Systems, B. W. Boehm, (September 70, R-520-NASA).

38. Experience with the Extendable Computer System Simulator, D. W. Kosy, (December 70, R-560-NASA/PR).

39. Computer Performance Analysis: Measurement Objectives and Tools, T. E. Bell, (February 71, R-584-NASA/PR).

40. Computer Performance Analysis: Applications of Accounting Data, R. A. Watson, (May 71, R-573-NASA/PR).

41. Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort, T. E. Bell, B. W. Beohm, and R. A. Watson, (November 72, R-549-1-PR).

SHARE

42. Study of OS 360 MVT System's CPU Timing, L. H. Zunich, SHARE Computer Measurement and Evaluation Newsletter, (February 7, 70, no. 3).

43. Computer Measurement and Evaluation Committee Reports, Thomas E. Bell, Chairman, (SHARE XXXIV Proceedings, March 70, vol. 1, pp. 492-547).

44. Systems Measurement--Theory and Practice, K. W. Kolence, (SHARE (SHARE XXXIV Proceedings, March 70, vol. 1, pp. 510-521).

45. Hardware Versus Software Monitors, T. Y. Johnston, (SHARE XXXIV Proceedings, March 70, vol. 1, pp. 523-547).

SYSTEMS DEVELOPMENT CORPORATION REPORTS

46. Two Approaches for Measuring the Performance of Time-Sharing Systems, Arnold D. Karush, (May 69, SP-3364).

THE COMPUTER BULLETIN

47. A Review and Comparison of Certain Methods of Computer Performance Evaluation, J. A. Smith, (May 68, pp. 13-18).

UNIVERSITY REPORTS

48. Evaluation of Time-Sharing Systems, Georgia Institute of Technology, (GITIS-69-72).

49. Performance Evaluation and Scheduling of Computers, Engineering Summer Conference, (July 1972, University of Michigan).

50. Statistical Computer Performance Evaluation, W. F. Freiberger, (1972, University of Rhode Island).

BOOKS

51. Joslin, E. O., Computer Selection. Addison-Wesley Publishing Co. (1968).

USE OF SMF DATA FOR PERFORMANCE ANALYSIS AND RESOURCE ACCOUNTING ON IBM LARGE-SCALE COMPUTERS

R. E. Betz

Boeing Computer Services, Inc.

INTRODUCTION

The search for ways to measure the performance of computer systems has led to the development of sophisticated hardware and software monitoring techniques. These tools provide visibility of the utilization of resources, such as CPU, channels and devices and also, with proper analysis, an indication of internal bottlenecks within the computer system. By themselves, however, these utilization values tell a data processing manager very little about his current capacity, how much and what type of additional work can be processed or whether the configuration can be reduced without suffering throughput degradation. The manager has a further problem in knowing when to apply the monitoring tools, in determining beforehand that the system is operating inefficiently or in identifying those applications which are causing the problem.

Boeing Computer Services, Inc. (BCS), when faced with the problem of managing many nationwide large scale IBM computers, turned to the use of System Management Facility (SMF) accounting data. Through experience and slight modifications to SMF a reporting system was developed that quantifies performance and capacity. This reporting system is known as the SARA(System Analysis and Resource Accounting) system and is in use in all computing centers managed by BCS.

This paper briefly describes the SARA system, discusses some of its uses in past studies, and discusses the impact of virtual systems on SMF reporting. The SMF data base is explained in order to put the SARA function in perspective with the other monitoring techniques.

SYSTEM OVERVIEW

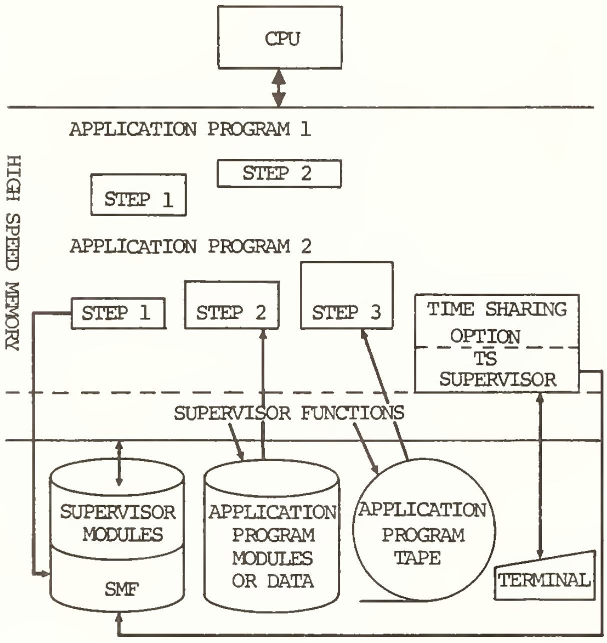
Figure 1 is a simple overview of a large scale BCS computer system, OS/MVT with TSO. As application programs are processed they make demands on the supervisor in the form of supervisor calls (SVC's) such as obtain core, allocate devices, execute channel program (EXCP) and set timer. Modules to perform these functions either are already resident in memory and are executed or are loaded from the system device and executed. Other portions of the supervisor read the job card deck; schedule, initiate, and terminate the job; and print the output.

The capability to perform online dataset editing and online application program execution is provided via the Time Sharing Option (TSO). TSO is a special type of application program which is given the ability to execute some privileged instructions usually restricted to the supervisor.

The System Management Facility is so named because it is a series of system exit points that allow the user to program decisions peculiar to his system. For example, an exit is provided immediately after a JOB CARD is read. The computing facility may have decided to abort all jobs with invalid account numbers and make a check of the job's account number at this exit point. When an invalid account number is encountered, an appropriate message is written and the job is terminated.

As the application program is processed SMF data is collected and stored in memory buffers. Wall clock time data is collected on such items as when the job was read in, when the job step was initiated, when device allocation started, and when the job step was terminated. Resource utilization data is collected such as how much memory was allocated, location and type of data sets or files allocated, and number of I/O accesses (EXCP's) made to each data set. When the job step terminates, the SMF data collected on it is output to a file. At the completion of all job steps for a job, another type of record is written to the file to describe the job. Similar data is collected causing different record types to be written at the completion of printing or of a TSO terminal session. All types of records are identified by a unique number (0 through 255) contained in the record. Note that the supervisor does not collect data on the functions it must perform in support of the application programs and thus SMF data is only indicative of resources used for application programs.

FIGURE 1.
SYSTEM OVERVIEW, OS/MVT WITH TSO



MONITORING TECHNIQUES

Software monitors are usually used to intercept supervisor activity (SVC's) for module loading activity, to sample internal tables for various queue lengths/times, and/or to sample devices for device busy time.

Hardware monitors physically attach to the computer system. They usually measure resource utilizations by timing the duration of pulses of the internal circuitry.

The disadvantages of the above monitors are that they tend to be expensive if not needed continually and that it is difficult to correlate the workload impact on the system. When a computer manager wants to know what his present performance is and what additional workload can be processed, it means very little to him that the channels are being used 30% and the CPU 55%.

Accounting data usually includes devices allocated, CPU time, wall clock time of critical events and memory allocated. This type of data can provide information on the work mix and on the resource utilizations of the application programs. The data has further advantages of (1) continual availability and (2) reporting in terms familiar to the computer manager.

BCS EXTENSIONS TO SMF DATA

BCS began using SMF for performance reporting as soon as the data became available with IBM system releases. As weaknesses of data content became apparent, programs were written by BCS for various SMF exits and modifications were made to internal system code to supplement the data. This additional data included the start time of device allocation and the start time of problem program loading (this data is present in the latest system releases). It also included counts of tape mounts due to end of volumes, counts of disk mounts, time in, roll-out status, and absolute memory address of region allocation.

Two significant additions to the SMF data were a calculation of the job step single stream run time, called Resource Utilization Time (RUT), and a calculation of the proportionate resources used by the job step, called Computer Resource Units (CRU's).

The RUT and CRU concepts were originated to provide repeatable billing to customers. A job can be processed on any BCS computer system and in any workload environment and be billed the same cost. RUT and CRU soon became important as performance indicators as discussed in the following sections.

THE SARA SYSTEM

The BCS entries of RUT and CRU and the optional SMF extensions allow the SARA system to overcome the limitations of accounting data for system analysis. In addition, during the processing of SMF data, SARA provides the capability of calibrating the data to the computer configuration and workload so that EXCP's can be transformed into device and channel utilizations. The results of SMF data are now calibrated to unique computing environments and can be different for device locations as well as device types. Although the device/channel utilizations are estimated, they help to pinpoint trouble spots much more readily than do counts of EXCP's. The calibration constants of each BCS computer system are calibrated periodically.

During the report program processing, the SMF data is analyzed by the machine states of the multiprogramming environment. The CPU and I/O activities of a job step are averaged over the period from program load time to job step termination. A snapshot of system processing at any time period would show a level of activity with a number of jobs processing at a specific CPU and I/O activity level with a specific amount of memory and devices allocated. These parameters represent a machine state until one of the job steps stops or another starts. The activities of all machine states are accounted and reported as illustrated in a later section which shows some of the report types.

Using CRU's, one can calculate CRU's per hour (the CRU's generated per hour of system active time) or the problem programs CRU rate (the CRU's generated per hour of problem program run time). These figures quantify the machine capacity required to process the given workload. CRU can thus be used as a figure of merit for computer performance.

Using RUT, one can calculate RUT hours per system active hour (this is termed "throughput" in SARA and is the accumulated RUT of all job steps over an interval of system active time). One can also calculate a job-lengthening-factor by dividing job step RUT into the actual job step run time. These figures indicate internal conflicts caused by poor machine loading or a poorly tuned system, and thus give an indication of multiprogramming efficiency.

SARA also calculates "memory wait time" and "device wait time". Memory wait time is the time from termination of the last job step to start of device allocation for the current job step. (This wait time also includes some device deallocation and initiate time, but long periods indicate a wait for memory.) Device wait time is the time from start of device allocation to start of problem program load.

TYPICAL SARA STUDIES

BCS makes a technical audit of its large scale computers at least once a year. The audit is performed by personnel who are familiar with hardware/software monitors and SARA. The audit establishes the CRU capacity of the machine, its performance in meeting priority demands, identifies bottlenecks and makes recommendations for system and/or configuration changes. The following sample reports were taken from audits and are presented not because of major benefits or savings that resulted, but because they demonstrate some of the uses of SMF type data when organized properly.

JOB SUMMARY REPORT

Before proceeding to a system study, it may be well to introduce the type of information provided by SARA. The JOB SUMMARY REPORT, shown in Figure 2, gives an overview of system performance and shows total job resource requirements. Figure 2 summarizes an eleven hour, second shift period of operation for a 370/165, although the report period is selectable and may cover any number of hours, days and/or weeks.

This particular report (Figure 2) came from the main processor of a dual 370/165 configuration driven by a LASP operating system. The 24002 (seven-track tapes) and the 24003 (nine-track tapes) device types are unique to this facility, since they were so named during the calibration input phase of SARA (the device type notation extracted from the operating system by SMF, denotes that a tape drive is a 2400 (or 3400) device type and makes no distinction between different types of tape drives).

This system is displaying relatively good performance with good "throughput" and "CRU/hour" values. Two criteria which indicate possible performance improvements are the relatively low "percent CPU" and "job lengthening factor". Memory size does not appear to be a problem per the low value of "memory wait" but "device wait" could be excessive. This system will be studied further in the following examples of SARA reports.

FIGURE 2
SARA JOB SUMMARY REPORT

SARA JOB SUMMARY REPORT FROM 73118 TO 73118
TOTALING 1 DAY MACHINE=165A

	NO. JOBS	STEPS	MEMORY WAIT	DEVICE WAIT
TOTALS	117	975	2.000	6.194

	CPU HOURS	RUN HOURS	RUT HOURS	CRU
TOTALS	4.945	50.896	35.189	6488.25

	24003 HOURS	24003 NUMBER	24002 HOURS	24002 NUMBER
TOTALS	9.565	499	4.992	306

	3330 HOURS	3330 NUMBER	2314 HOURS	2314 NUMBER
TOTALS	4.967	2778	0.197	103

THROUGHPUT = 3.20
AVERAGE CRU/HOUR = 589.75 LENGTHENING FACTOR
OF AVERAGE JOB = 1.45 PERCENT CPU = 44.9
UP TIME = 11.002 TOTAL TIME ACCOUNTED FOR=11.002
NUMBER OF TAPE MOUNTS = 861, NUMBER OF DISK
MOUNTS = 7

RESOURCE ALLOCATION DISTRIBUTION

To effectively manage a configuration, it is necessary to know the utilization of resources and specifically how that utilization is distributed. To configure the proper number of tape drives, one must know how much time the minimum number and subsequent numbers of tape drives are required. The "Resource Allocation Distribution" report, shown in Figure 3, illustrates this distributive data.

As the multiprogramming machine states are analyzed, SARA records the discrete number or amount of resources allocated or used at different machine states. Figure 3 is an example of this type of data for allocated nine-track tape drives. The report period was based on a predominantly scheduled production period when tape drive demand was the highest. The 24003 device type is unique to this facility since it was so named during the calibration input phase. The report shows:

- The discrete increments of the resource (in this example zero to twenty eight 24003 tape drives)
- The time in hours that exactly 0, 1, 2, ..., 28 tape drives were allocated
- The percent of the total time period that is represented by the allocation time of the discrete increment
- The accumulated percent
- A graphic distribution to the nearest one percent of the discrete increment percentage.

The arithmetic mean number of tape drives allocated is reported as well as the "number of units or less" that are allocated 95% of the time.

This particular distribution is from the same main processor of a dual 370/165 configuration as was discussed in Figure 2. The total configuration had 84 tape drives, 28 nine-track and 20 seven-track on the main processor and 20 nine-track and 16 seven-track on the support processor. In addition, the configuration had thirty two 3330 disk spindles and sixteen 2314 disk spindles. Facility management wanted to decrease their dependence on tape and add more 3330 modules but they were constrained to maintain the same configuration costs with no schedule slides. A logical tradeoff was: How many tape drives can be released to recoup the costs of additional 3330 modules?

The Resource Distribution Reports showed that four 9-track drives (with minimal risk) and no 7-track drives could be released from the main processor. No 9-track drives but four 7-track drives could be released from the support processor. To quantify the risk, the eight drives were "taped off" and restricted from operations use. "Before" and "after" runs of SARA showed no discernible degradation of capacity or throughput and resulted in a trade of tape drives for disk.

These distribution reports are available for other devices, memory, CPU, channels, concurrent job streams, throughput, CRU's and direct access data sets.

FIGURE 3
RESOURCE ALLOCATION DISTRIBUTION

```
SARA RESOURCE DISTRIBUTION REPORT FROM 73118
TO 73118 TOTALING 1 DAY  MACHINE=165A

ALOC              ACCUM  DISTRIBUTION
24003 TIME  %TIME   %     *   10   *   20 * 30 * 40

    0 0.002  0.02   0.02
    1 0.003  0.03   0.04
    2 0.005  0.04   0.08
    3 0.018  0.16   0.24
    4 0.048  0.44   0.68
    5 0.359  3.27   3.95  AAA
    6 0.179  1.63   5.58  AA
    7 0.195  1.77   7.35  AA
    8 0.464  4.22  11.58  AAAA
    9 0.417  3.79  15.37  AAAA
   10 0.637  5.79  21.16  AAAAAA
   11 0.212  1.93  23.08  AA
   12 0.810  7.36  30.44  AAAAAAA
   13 0.980  8.91  39.35  AAAAAAAAA
   14 0.479  4.36  43.71  AAAA
   15 0.836  7.59  51.30  AAAAAAAA
   16 0.815  7.41  58.71  AAAAAAA
   17 0.358  3.26  61.97  AAA
   18 0.905  8.23  70.20  AAAAAAAA
   19 0.789  7.17  77.37  AAAAAAA
   20 0.256  2.33  79.70  AA
   21 0.536  4.87  84.57  AAAAA
   22 0.497  4.52  89.09  AAAAA
   23 0.575  5.23  94.32  AAAAA
   24 0.097  0.88  95.20  A
   25 0.232  2.11  97.30  AA
   26 0.165  1.50  98.80  A
   27 0.071  0.65  99.45  A
   28 0.061  0.55 100.00  A

   MEAN UTILIZATION = 15.39

   95% UTILIZATION UNDER 23.78
```

JOB STREAM EFFICIENCY REPORT

A question frequently asked of an OS/MVT environment is: What is an optimum number of active concurrent jobs? Too many active jobs may produce internal conflicts sufficient to degrade capacity compared to a number of smaller jobs. Too few active jobs may not achieve maximum capacity.

As concurrent job streams are determined from the multiprogramming machine state analysis, SARA records the average number or amount of resources allocated or used and the performance criteria at the different levels of concurrent job streams. Figure 4 is an abbreviated example of this type of data showing only the performance related criteria.

The report shows for each level of concurrent job streams:

- the level of job streams
- the percent of time that exactly 0, 1, 2, ..., 7 job streams were active
- the throughput
- CPU percent
- CRU's/hour

The SMF data was collected from the same dual 370/165 configuration as in Figure 3 but for a period representing more of a demand (as opposed to scheduled) workload.

This facility was previously running ten initiators on each system. The BCS software monitor showed heavy device and channel queuing and so the active initiators was reduced to seven. The report in Figure 4 shows vividly that neither ten nor seven active initiators was correct.

The support processor that must handle the spooling activity (reading cards and printing output) and scheduling of jobs seems to have bogged down at more than 4 job streams as witnessed by the throughput and CRU's/hour (note that data points at less than 10% of the active time are usually discarded as too small a sample). In addition, LASP response to Remote Job Entry (RJE) and a BCS online system seems to have slowed down drastically during the demand workload periods when short jobs were running and much initiating, terminating, card reading and printing output was occurring. This was verified by timing the duration for LASP to refurnish the RJE print buffers and timing its response time to the online/LASP interface commands such as status of job, backlog, submit, etc. The support processor while providing OS services for a large number of job streams and LASP support for both systems, generated internal supervisor queuing such that the problem programs were seriously degraded.

The main processor, however, had not yet peaked at seven initiators and continued to produce more work with each increase of concurrent jobs. It is possible that an increase in initiators would produce an increase in throughput and CRU's per hour.

It was recommended that the number of initiators on the support processor be reduced by one and on the main processor be increased by one. Although the recommendation was followed, the results of the changes are not yet available.

FIGURE 4
JOB STREAM EFFICIENCY REPORT,
DUAL 370/165's

SUPPORT PROCESSOR

Concurrent Job Streams	Time	Thruput	% CPU	CRU's/Hr.
1	7	.81	16	284
2	6	1.66	30	482
3	20	2.66	48	651
4	23	3.35	49	581
5	29	2.77	35	457
6	13	3.21	26	448
7	2	4.09	20	432

MAIN PROCESSOR

Concurrent Job Streams	Time	Thruput	% CPU	CRU's/Hr.
1	0	0	0	0
2	0	0	0	0
3	2	2.84	39	447
4	7	3.06	46	484
5	13	3.96	45	567
6	35	4.09	47	574
7	43	4.55	49	619

INTERVAL AVERAGE REPORT

Some facilities have a cyclic workload, i.e., the daytime workmix characteristics are different from the overnight workmix characteristics. To analyze these systems, a capability is required to observe performance at different intervals of the day, perhaps hourly or by shift. The "Interval Average" report provides this capability.

The interval average report shows chronological intervals of average activity throughout the report period. Figure 5 is an abbreviated example of this report showing problem program hourly averages of CPU per cent, CRU's/hour, throughput, core allocated, number of job streams and calculated channel utilization for the time period from 0800 to 1800.

This data was collected from a 370/165 running HASP RJE and BCS TSO. The workload is primarily priority demand jobs which are submitted via RJE with up to 20 communication lines and TSO supporting 60 to 70 users. As jobs are submitted, a "job class" is calculated and assigned by the system based on job resource requirements.

Initiators are given job class assignments to best satisfy the priority demands (e.g. an initiator assigned job classes A, B, C must process all jobs of class A before B or C; if no class A jobs are in the queue, it may process class B; if no class A's or B's, it may process class C). The calculated job classes and the initiator class assignments are such that small jobs are given preferential treatment. The larger the job, the longer the committed time to process the job.

This facility had been experiencing unacceptable TSO response at 70 users. To reduce internal conflicts and/or TSO degradation, the number of active initiators was reduced from six to four during peak TSO periods from 0800 to 1800. Thus, four problem programs would be the most that could run concurrently. The two deducted initiators had been primarily assigned to processing large jobs.

The reduction in active initiators did result in reduced TSO response time. SARA showed a reduced job lengthening factor, and the BCS software monitor showed a significant reduction in device/channel queuing. There was also a reduction in processing CRU's/hour.

Figure 5 is typical of processing with four active initiators. It is apparent from the Interval Average Report that:

- the job class assignments were far too weighted to small jobs as witnessed by the poor core utilization which in turn produced few CRU's
- the input workload could not keep the initiators active with their present class assignments per the low number of job streams
- the deducted initiators should be activated at 1600 per the reduction in job streams (this is about the time that terminal users quit submitting small jobs since they won't be processed by quitting time and submit large overnight jobs; this is also the time of highest job backlog).

The initiator job class assignments were reworked (this is a separate study in itself but relies heavily on job class and job priority resource requirements as reported by SARA). The result was an increase in CRU's/hour generation from less than 300 to 350, better satisfaction of priority demands and little effect on the improved TSO response times. The system with four initiators was doing better than the previous operation with six initiators (although the new job class assignments were primarily responsible, a concurrent effort to reorganize the system packs was also beneficial).

FIGURE 5
SARA INTERVAL AVERAGE REPORT

TIME INTERVAL	PROB CPU	CRU's /Hr.	THRU PUT	CORE ALLOC
0800-0900	6	142	1.23	198
1000	19	365	3.79	524
1100	8	195	2.11	323
1200	14	267	2.24	392
1300	31	351	3.01	347
1400	21	350	2.90	402
1500	17	251	2.24	402
1600	11	234	1.96	351
1700	8	257	2.30	335
1800	36	340	2.55	246

JOB STMS	CHAN 2	3	4	5
1	20	9	4	4
4	26	2	8	2
3	10	6	5	7
3	36	27	8	15
3	26	1	3	0
3	59	3	3	25
3	22	4	10	11
3	18	8	27	10
2	14	3	47	0
2	16	5	3	11

THE FOLLOW-UP

The foregoing studies resulted in recommendations for system improvements, but a study should not end at that point. It is relatively easy to obtain a tape of SMF data for post change analysis even at an offsite location. Analysis of the results of change not only allows further tuning of the original recommendation but provides experience in the most significant types of changes.

Also, the studies make mention of throughput, CRU's per hour and the job lengthening factor as performance indicators. With experience and a variable number of computer configurations to investigate, BCS performance analysts have come to recognize the performance capability of a given machine configuration. This allows an almost instantaneous analysis of a "machine in trouble" and if the problem cannot be found, the hardware/software monitors are used for a thorough analysis.

VIRTUAL MEMORY SYSTEM

Virtual memory systems provide up to 16 megabytes of virtual core, a faster and larger real memory, and a more integrated operating system with internal processing algorithms that can be tailored to the workload. The concept of job processing does not change. The following discussion is based on OS/VS2-Release 1.6 unless specific details are known on Release 2.0. For example, Release 2 offers a System Activity Measurement Facility (MF1), a monitoring function that sounds like a limited software monitor, but little is known about it at present.

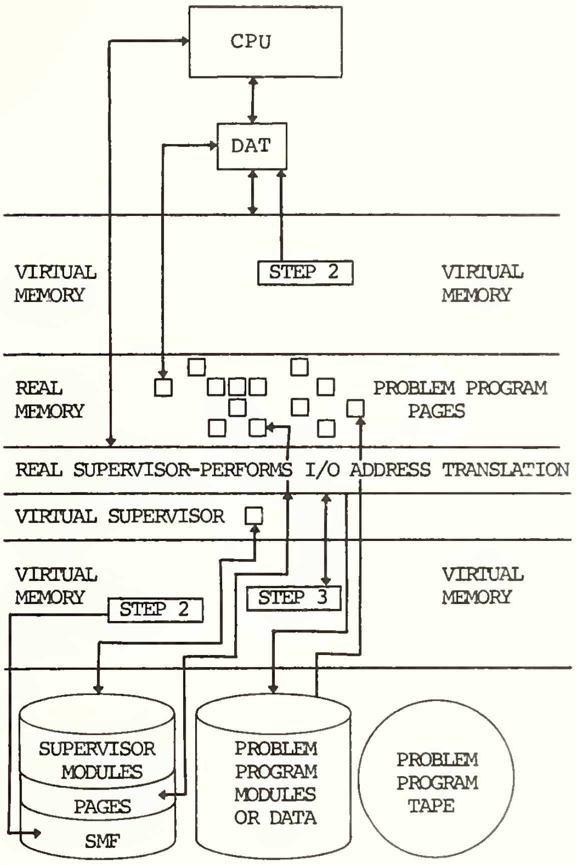
Figure 6 is a simplified overview of the OS/VS2 system operation. The primary functional features and differences over OS/MVT are:

- the Dynamic Address Translator (DAT), a hardware device that translates CPU "virtual memory" accesses into "real memory" address locations
- the IO supervisor additional software function which translates IO virtual memory addresses into real memory address locations
- virtual memory allocation to job steps in 64K byte blocks
- real memory utilization by job steps in 4K byte pages.

The virtual system philosophy then is to increase the user's memory size (a bottleneck heretofore) by giving him up to 16 megabytes of usable core and to decrease the real memory needed by a particular job by making the job's most active 4K pages reside in real memory. This along with the internal priority schemes should allow the user to load up the system with jobs and let the system work them off as efficiently as possible.

Under OS/VS2, SMF records for job steps and TSO sessions include page ins and page outs. In addition, for TSO, the number of swaps, swap page ins and swap page outs are recorded. A system type SMF record gives counts of system page ins, page outs, page reclaims and TSO region swapping statistics.

FIGURE 6
SYSTEM OVERVIEW, OS/VS2



VS2 OR MVT

The decision to change from a basic MVT system (i.e. 370/155,165) to a VS2 paging system (i.e. 370/158,168) involves many considerations including cost, performance and/or special applications. The cost tradeoffs are well defined but one must still determine what performance gains the new, faster memory will produce, what paging overhead will be suffered and what the performance impact will be on applications programs and/or time sharing. Before proceeding with replacement, BCS chose to benchmark the new systems.

Benchmark jobs were obtained from each BCS large scale computer. The BCS TSO stimulator was rewritten for OS/VS2 (the TSO stimulator called TIOS, TSO Input Output Simulator, emulates the TSO user's activity per predefined scripts). The BCS software monitor was rewritten for OS/VS2. Where possible, tests were conducted using the hardware monitor. SARA was modified to incorporate the new paging data.

Testing was done at IBM data centers on both VS2 and MVT.

APPLICATION PROGRAMS BENCHMARK RESULTS

Figure 7 shows benchmark results of application programs (jobs) run under OS/MVT and VS2 on a 370/158, 2 megabyte system and a special case of VS2 on a 370/158, 1 megabyte system. This particular benchmark contains 56 jobs and comes closest to representing an overall BCS workload although other benchmarks of CPU and I/O boundness were also run. The test results are somewhat tailored to BCS test requirements (such as for initiators with MVT) and BCS fine tuning. The test of the 1 megabyte system was prematurely aborted at 75% completion because excessive paging extended the elapsed time beyond the scheduled time.

The test data show some interesting results. The CRU generation and throughput rates of MVT and VS2 under these test environments were about equal. The job lengthening factor was very much less under MVT and this was an objective of that test. The CPU accounted to application programs, as shown by SARA, was less under the VS2 system. The total CPU, as shown by the hardware monitor, was less under the MVT system. This apparent CPU contradiction is a result of IBM's decision to delete from SMF accounting those routines that produced variable CPU times under variable paging requirements.

The ratio of SMF CPU% to total CPU% on the MVT test results is typical of past MVT system studies. The lower channel hours required under VS2 are reflective of very little paging activity.

The VS2,1 and 2 megabyte systems test results make another interesting comparison. The excessive job lengthening factor of 4.10 indicates inefficient job processing for the 1 megabyte system. This system required 260,000 pages as compared to 1800 pages for the 2 megabyte system. Approximately 40% of the paging was performed for two jobs. Although the channel hours are about the same, much of the channel time for the 1 megabyte system was spent in paging. The excessive paging shows increased CPU overhead (as evidenced by the increased ratio of supervisor CPU to total CPU) and some time waiting for pages (as evidenced by lower total CPU than under the 2 megabyte system) and results in inefficient job processing. Note that the thrashing threshold algorithm which is supposed to limit paging was not available during these tests.

FIGURE 7
370/158 TEST RESULTS

	OS/MVT(21.6) 2 MEGABYTE 4 INITIATORS	VS2 (1.6) 2 MEGABYTE 7 INITIATORS	VS2 (1.6) 1 MEGABYTE 7 INITIATORS
1.ELAPSED TIME HRS (DATA FROM SARA)	1.907	1.893	1.916*
2.CRU's/HR (SARA)	311	306	230
3.THROUGHPUT (SARA)	2.03	2.03	1.56
4.JOB-LENGTHENING FACTOR(SARA)	1.60	2.70	4.10
5.PROGRAM CPU% (SARA)	64	57	44
6.TOTAL CPU % (HDWE MONITOR)	75	88	75
7.PROBLEM STATE/ SUPERVISOR STATE-CPU % (HDWE MONITOR)	54/21	55/33	38/37
8.CHANNEL HRS (HDWE MONITOR)	1.57	1.47	1.48

* TEST ABORTED PREMATURELY

TSO BENCHMARK RESULTS

TSO was tested using the BCS written TIOS (TSO Input Output Simulator). TIOS simulates any number of terminal users each performing a set of unique commands per a pre-established script. The script requirements were developed based on a survey of experienced BCS terminal users. TIOS allows simulation of a proportionate TSO user load across different increments of users and thus provides a consistent basis of analysis (quite different from analysis of the production TSO system with its random user load).

Figure 8 shows an abbreviated SARA interval report (much like Figure 5 for application programs) depicting TSO activity only. The variables that were important in analyzing system performance for application programs are also important for analyzing TSO. This report shows the number of users who started their session during the interval, the average number of users logged on, the percent of time all users actually occupied memory (i.e. were swapped in), CPU percent, throughput and CRU's per hour. A thorough analysis of TSO can be made by using SARA type data, response time data provided by the analysis program of TIOS, and hardware/ software monitor data.

In general, the test results show that TSO/VS2 at the present time cannot compete with TSO/MVT. TSO/VS2 generates much paging activity, uses three times as much CPU as MVT (e.g., from 16% under MVT to 52% under VS2), produces 30% less CRU's per hour, and gives longer response times. These observations are valid for as few as 15 users to as many as 60 users.

TEST RESULT CONCLUSIONS

The test results showed that the VS2 paging system is not yet an acceptable replacement for the MVT system but is sufficient to handle some of the special applications within BCS (e.g., a large memory on line system). The excessive overhead will undoubtedly be reduced in later system releases. These releases will warrant testing because of the operating system advantages e.g., job scheduling, device allocation, priority grouping.

Other test results showed that with either operating system:

- The 370/158 produced better performance than the 370/155
- the software translation of virtual addresses in a strictly I/O bound benchmark produced more overhead on 370/158 than was balanced by the improvement in CPU speed over the 370/155
- Automatic Priority Grouping does produce better performance.

FIGURE 8
SARA TSO INTERVAL REPORT

START TIME	USER STRT	NO. USER	% M STOR	% CPU	THRU -PUT	CRU /HR
20.750	0	0.0	0	0	0.0	0
21.000	0	0.0	0	0	0.0	0
21.250	0	0.0	0	0	0.0	0
21.500	44	17.8	20	4	0.04	20
21.750	0	45.0	179	36	0.29	48
22.000	1	44.8	175	33	0.27	46
22.250	1	45.0	174	32	0.27	46
22.500	0	45.0	174	32	0.27	47
22.750	0	45.0	174	32	0.27	47
23.000	0	38.1	141	24	0.22	39
23.250	0	0.0	0	0	0.0	0
23.500	0	0.0	0	0	0.0	0

IMPLICATIONS OF VS2 ON SARA

The test results show that excessive paging produces system degradation. This means that paging activity and performance relative to the paging activity will somehow have to be monitored. This also holds true for the priority/scheduling schemes and the trashing threshold setting.

SARA now contains paging activity for application programs and for the total system. This data, at a minimum, allows one to recognize specific application programs that require excessive paging (and should be reorganized) and to establish performance levels at different paging levels. With experience, it is hoped that performance correlations can be found that will evolve into more concise parameters depicting overall system performance.

As new SMF data becomes available with new OS/VS2 releases, notably MF/1 under Release 2, SARA should have the flexibility to accommodate the data incompatibility with its reporting requirements. For example, MF/1 data may replace our current calibration function or if overhead to collect MF/1 data is high, it may be used infrequently to support the adjustment of the calibration constants.

SUMMARY

BCS has found that the use of accounting data provides a relatively quick and flexible method of system analysis and resource accounting. With experience and appropriate modifications to SMF data, BCS has been able to establish capacities, determine more cost effective configurations and resolve workload scheduling conflicts for large scale computers.

Incorporating performance indicators into the SARA reporting system not only aids in evaluating the current production systems, but also aids in predicting the capability of new systems.

Recently announced operating system changes with their associated scheduling and priority parameters will make more important the quantitative measurement of work done and will increase the value of SARA in studying these systems.

BIBLIOGRAPHY

1. BCS Document No. 10058, SARA System User's Guide

2. BCS Document No. 10068, 370/158 Advanced Technology Testing

3. BCS Document No. 10077, TIOS Users Guide

4. BCS Memo, Technical Audit of Northwest District 370/165 A&B

5. BCS Memo, Technical Report - Data Services District 370/165 Evaluation

6. IBM Publications GC35-0004-OS/VS SMF, GC-6712-OS SMF, GC28-0667-OS/VS2 Planning Guide for Release 2.

USING SMF AND TFLOW FOR PERFORMANCE ENHANCEMENT

J. M. Graves

U.S. Army Management Systems Support Agency

Agency Overview

Before I get too involved in this presentation of how we use SMF and TFLOW for performance enhancement, let me give you a few words of background on my Agency, USAMSSA, The U. S. Army Management Systems Support Agency. We were created several years ago to serve the Army Staff's Data Processing requirements. From our site in the Pentagon, we handle the full range of commercial and scientific applications, with a slight bias toward the commercial. Applications are processed via batch, TSO, and RJE on a 3 megabyte IBM 360/65 and 1 and 1/2 megabyte IBM 360/50 using OS/MVT and HASP. As you might expect, we have a large number of peripheral devices attached to the two machines, some of which are shared. We have a number of operational conventions, but I need mention here only those two which have an impact on systems performance. The first operating convention is what I call the SYSDATA convention, whereby all temporary disk data sets are created on permanently mounted work packs given the generic name of SYSDATA at System's Generation Time. The second operating convention to bear in mind is the fact that of the total of 80 spindles on the two machines, only 41 are available for mounting our 150 private mountable packs, the remainder being used by permanently resident packs.

Problem Areas Defined

We have in the past made use of a number of tools to analyze system performance: two hardware monitors - DYNAPROBE and XRAY, a software monitor CUE, and last but not least, personal observation the computer room. These tools are all excellent in defining those areas of system performance that need optimizing, e.g., the CPU is not 100% busy, the channels are not uniformly busy at a high level, certain disk packs show more head movement than others, and there are an excessive number of private pack mounts. However, none of the usual tools tell you what to do to eliminate or alleviate defined system bottlenecks. Our approach to filling this information gap, is to use free IBM software - SMF and TFLOW - to produce data for subsequent reduction into reports which at least point you in the right direction. SMF is a SYSGEN option and TFLOW, which records everything that goes through the trace table, is available from your IBM FE as a PTF. After making the indicated system changes, we go back to the monitors to validate results.

Core Availability

The first area of system performance we examined was core availability, which on most machines, is one of the most important constraints on the degree of multiprogramming, and therefore on the maximum utilization of the machine. The more tasks that are in core executing concurrently, the higher will be the utilization of the CPU, the channels, and the devices, provided of course that device allocation problems do not arise from the additional concurrently executing tasks. If allocation problems do occur as a result of an increased level of multiprogramming, a rigidly enforced resource oriented class structure will help. If this approach fails to reduce allocation lockouts to an acceptable level, device procurement is about the only other avenue of approach. One way of justifying (or perhaps I should say rationalizing) the acquisition of additional peripheral devices to support the additional tasks now multiprogramming is to regard the CPU cost and the core cost per additional task as essentially zero, since these costs were a fixed item <u>before</u> you began system tuning. In summary, by making more core available, one should be able to support extra applications at no cost, or in the worst case, the cost of enough new devices to lower allocation contention to an acceptable level. Short of bolting on more core, there are a number of approaches one can adopt to increase the average amount of core available at any one time, and I will describe two of our approaches.

The Weighted Core Variance Report

In our computing environment, since we do not bill for resources consumed, there is no economic incentive for a user to request only that amount of core which he actually needs to execute his program. To the contrary, many considerations in the applications programmer's mileau influence his REGION request to be on the high side. First, he wants to avoid S 80A abends-core required exceeds core requested. Second, he wants to provide enough core over and above the requirements of his program to provide a dump in the event of an abend. Next, he wants to provide room for future expansion of his program without the necessity of also changing his JCL. This consideration is particularly important to him if his JCL has been stored on the catalogued procedure library, SYS1.PROCLIB. Next, there is the certain knowledge that coding and keypunching effort can be saved by specifying the REGION request at a higher level than the step EXEC statement, e.g., the EXEC PROC statement, or on the JOB statement. In either case, the REGION specified is applied to each step of a group of steps, and must therefore be large enough to enable the largest step of the group to execute. Obviously this practice can lead to gross core waste if there is a large variance between the REGION requirement of the largest and smallest steps. For example, we have experienced 300K compile, link, and go jobs when the 300K was required only for the go step.

One approach to this problem would have been to introduce a JCL scan to prohibit the use of the REGION parameter on the JOB statement or the EXEC PROC statement. I personally preferred this approach, but it was the consensus of our management that there was nothing wrong per se with specifying the REGION requirement at a higher JCL level than the STEP statement. Only when such a specification results in wasted core should the user be critisized. Since there is no way to know how much core actually will be used until after it is used, and after the fact reporting system had to be developed to identify daily, on an exception basis, those users whose wasteful REGION requests were causing the greatest impact on core availability. This mechanism is the Weighted Core Variance Report. The idea of the report is to produce an index number for each job step whose magnitude is a measure of the degree of waste his region request has produced in thd system. Acceptance of the report by management was contingent upon the repeatability of the computed index number. From SMF type 4 step termination records, the variance between the requested core and the used core is computed, a 6K abend/growth allowance is subtracted, and the result is weighted by the amount of CPU time used by the step. This product is further weighted by dividing by the relative speed of the hierarchy of core where executed. Negative results were not considered, since this result means the programmer was able to come within 6K on his core request. CPU time was used rather than thru time since it was felt that this figure was more independent of job mix considerations. The weighted variances are listed in descending order with a maximum of 20 job steps per administrative unit. It was felt that 20 job steps was about the maximum that a manager would attend to on one day. A minimum index value of 500 was established to eliminate reporting on trivial index numbers. The report is distributed daily to 40 or 50 management level personnel. Implementation of the report was met with widespread grousing which has continued up to the present. This, I feel, is a strong indication that the report is serving its purpose.

Access Method Module Utilization Report

Lest I give the impression that my idea of system tuning is to attack problem programming, let me describe the second of our approaches to achieve a higher level of core availability - One of the most interesting design features of OS is the use of re-entrant modules resident in a shareable area of core designated the Link Pack Area. If these modules were not so located and designed, there would have to be multiple copies of the same module in core at the same time - one copy for each task that required its services. By having these modules core resident, the various tasks multiprogramming can share the one copy, thus effectively saving an amount of core equal to the number of concurrently executing tasks, minus 1, times the size of the module. The higher the degree of multiprogramming and the larger the module in question, the greater is the benefit in core savings, or looking at the situation from another point of view, the greater is the core space availability. This increase in availability can be anything but trivial on a machine with a healthy compliment of core, since some access method modules are substantial in size - 2 to 4K for some BISAM & QISAM modules for example. Our criteria for validating the RAM list is to select for link pack residency only those modules which show concurrent usage by two or more tasks more than 50% of the time. Unfortunately, we were unable to locate any free software that would give us a report of concurrent utilization of access methods. We then looked to see if we could generate such a report from the data we had on hand. Such data would have to provide a time stamp giving the date and time that each task began using each access method module, and another time stamp when the task stopped using the module. If we had this information, it would be a relatively simple manner to calculate for each access method module the length of time there were from zero to 15 concurrent users and then relate the total time for each level of multiprogramming to the total measurement time as a percentage.

It becomes clear that the stop and start timestamps were already available in the form of SMF type 4 Step Termination records and the SMF type 34 TSO Step Termination records. What was not available was the all important access method module names invoked by each step. This is where TFLOW gets into the act. We had used TFLOW previously to get a more complete count of SVC requests than CUE supplies, and so were familiar with the fact that when SVC 8 (load) is detected, the module loaded was provided in a 32 byte TFLOW record. Therefore, in the TFLOW data, we had the access method module name, but no time stamps or step identification. The problem was how to relate the TFLOW data and the SMF data. Fortunately, TFLOW gives the address of the Task Control Block (TCB) of the task requesting the access method. This information is provided to an optional user-written exit at execution time. By searching a short chain of control blocks - from the TCB to the TCT to the JMR - we were able to locate the SMF JOBLOG field consisting of the JOBNAME, READER on date, READER on time. This JOBLOG data is reproduced in every SMF record produced by the job in execution, and is unique to that job. Also located in the JMR is the current step number. By copying the JOBLOG and step number to a TFLOW user record also containing the access method module name, we have a means of matching the TFLOW user record to the SMF step termination record written sometime after the access method module is loaded. By suppressing the output of the standard TFLOW record, and recording only those user records that represent a load request for an access method module, the overhead attendant with TFLOW's BSAM tape writing is kept to a minimum.

RAM List Validation Results

As a direct result of this analysis of access method module utilization at USAMSSA, we have recommended that the following changes be made to the standard IBM RAM list:

Additions:---11---Totaling-12044-bytes

Deletions:---19--- " --17538--bytes

Some of these additions and deletions were surprising. The additional BISAM and QISAM modules indicated a much heavier use of ISAM than we had imagined. The deletion of the backward read on tape, and the various unit record modules is surprising only in that we should have thought of it before but didn't, since practically everyone at USAMSSA uses disk sort work files instead of tape sort work files where backward reading is done, and having HASP in the system precludes a number of unit record operations, since HASP uses the EXCP level of I/O.

What Increased Core Availability has meant to USAMSSA

Use of these reports, plus other systems tuning techniques, has enabled us to go from 12 to 14 initiators in our night batch setup on the model 65, and from 3 to 5 TSO swap regions plus an extra background initiator in our daytime setup. CPU utilization has gone from the 65% - 75% busy range to 90 - 100% busy. Except for weekends, we are able to sustain this level of activity around the clock on the model 65.

So much for core availability, the increase of which was intended to tap the unused capacity of our CPU. To a large degree, we have succeeded in this area. However, the increased degree of multiprogramming has tended to exaggerate whatever I/O imbalances previously existed. We wanted to be able to keep a continuous eye on the disk usage area, since we had noted that some packs were busier than others and that some packs were frequently mounted and dismounted. We did not want to run Boole & Babbage's CUE every day, all day to get this information, for several reasons: systems overhead for one, and the uneasy feeling that maybe the required information is somewhere out in the SMF data set. We certainly were paying a considerable system overhead in having SMF in the first place; wouldn't it be nice if we got something more out of it than job accounting and the core availability uses I've already described.

Disk Multiprogramming Report

We developed a report, the Disk Concurrent Usage Report, which gives an approximation of the CUE device busy statistic. We started with the assumption that if there are a large number of concurrent users of a single disk pack, it is virtually certain that there will be a correspondingly high device busy statistic and that probably a large component of this busy figure is seek time. Also for private packs not permanently mounted, one can reasonably infer that a medium to high degree of concurrent use requires many operator interventions for mounts. Getting the measure of concurrent disk utilization follows the same approach as concurrent access method module utilization. Again type 4 SMF records were used to supply the start and end time stamps for each step run in the system. Instead of having to go to TFLOW to pick up the name of the shared resource, this time we were able to find the volume serial number (VOL/SER) of each disk pack containing each data set on which end of volume processing occurred in the SMF types 14 and 15, or EOV records. The types 4, 14, and 15 SMF records are matched to one another by the same common denominator as was used in the Access Method report -- the SMF JOBLOG, consisting of Jobname, Reader On Date, and Reader On Time. Locating the step which processed the data set which came to end of volume is accomplished by "fitting" the record write time of the type 14 or 15 record to the step start and end times. Once we have the time a step starts to use the pack and the time is stops using the packs, it is a simple matter to resequence the combined data and compute the percentage of time there were varying numbers of concurrent users on the pack.

We have used this report in several different ways. One way is to validate the permanently resident (PRESRES) list for private packs; that is, are all private packs that are permanently resident really busy enough to justify their mount status, and are there any nonresident packs which are so busy they would be good candidates for addition to the PRESRES list? Another way we've used this report is to check the activity of our SYSDATA, or work packs. Because OS bases its allocation of temporary data sets on the number of open DCB's per channel, one may wind up with considerably more activity on some work packs than on others. Our circumvention of this problem is to move a frequently allocated but low I/O activity data set from a relatively busy channel to the channel containing the high activity SYSDATA work pack. This data set is typically a popular load module library. A big problem in USAMSSA as far as system tuning goes is the dynamic nature of the use of private data sets as systems come and go. A data set that is very active one month may be very inactive the next month. The only way to keep up with the effects that this produces in the system is by constant monitoring and shuffling of data sets.

Data Set Activity by Disk Pack Report To aid us in getting a better idea of the nature or quality of the disk activity noted in the previous report, we produce a report which gives a history of the activity on each pack by data set name, the number of EOV's recorded and the DDNAME of the data set, where that information would be significant. We wanted to know which data sets were being used as load module libraries so that we would have good candidates for relocating to a channel containing an over-used SYSDATA pack. We also wanted to know which data sets were used by batch only, TSO only, and both batch and TSO in order to have the best group of data sets placed on the 2314 spindles which are shared between our two computers. All of the above information is contained in the SMF type 14 and 15 records, with the exception of the TSO versus batch breakout, which we accomplish by analysis of our highly structured jobname. Additionally, the number of mounts for the pack is assumed to be the number of dismounts, which are recorded in the type 19 SMF record. The number of IPL's and ZEOD's (operator Halt command) must be subtracted from this number if the pack was mounted at IPL time or ZEOD since a type 19 record is also written at these times.

We have also used this report to reorganize disk packs, particularly where one data set is the cause for most of the pack mounting. If the data set is very heavily used, we may try to find space for it on a permanently mounted pack.

Conclusion

We feel that the reports just described can be used as a vehicle for system tuning, in that they suggest courses of action by pointing out system anomalies. We can increase core availability by monitoring programmer core requests with the Weighted Core Variance Report and by modifying the standard IBM RAM list to accurately reflect USAMSSA's use of access methods. We can reduce disk I/O contention by monitoring disk utilization with the Disk Concurrent Usage Report and by monitoring data set activity with the Pack/DSN Activity Report. In a more passive sense, the reports can be used as a daily verification that the system stayed in tune without the attendant inconvenience of running a hardware or software montior. We find these reports to be useful. If you are interested in acquiring them, see me during the coffee break, or see Mert Batchelder and he will give you my address.

USACSC SOFTWARE COMPUTER SYSTEM PERFORMANCE MONITOR: SHERLOC

Philip Balcom and Gary Cranson

U.S. Army Computer Systems Command

1. ABSTRACT.

This technical paper describes the internal and external characteristics of SHERLOC, a 360 DOS software monitor written by the US Army Computer Systems Command. SHERLOC primarily displays 1) the number of times each core image module is loaded, and 2) CPU active and WAIT time, by core address, for the supervisor and each partition. The major advantage of SHERLOC over most, if not all, other software monitors is that it displays supervisor CPU time by core address. In this paper emphasis is placed on the concepts required for a knowledgable system programmer to develop his own tailor-made monitor. Results of using SHERLOC since completion of its first phase in August 1973 are discussed. SHERLOC stands for Something to Help Everybody Reduce Load On Computers.

2. GENERAL DESCRIPTION OF SHERLOC.

SHERLOC is started as a normal DOS job in any 40K partition. During initialization, SHERLOC requests information such as the sampling rate from the operator. When the operator then presses the external interrupt key, SHERLOC begins sampling at the specified rate, accumulating, in core, all samples. When the operator presses the external interrupt key a second time, SHERLOC stops sampling and allows the operator to indicate whether he wants SHERLOC to terminate, or to record the gathered data on tape or printer, or a combination of the above. After satisfying these requests, if SHERLOC has not been told to terminate, the operator can press external interrupt to cause SHERLOC to continue sampling without zeroing its internal buckets.

A separate program named HOLMES exists to print data from the tape or to combine data from the tape by addition or subtraction before printing. Thus, in monitoring for two hours, if data is written to tape at the end of each hour, a printout for the last hour alone can be obtained by running HOLMES and indicating that the first data is to be subtracted from the second. Naturally, HOLMES stands for Handy Off-Line Means of Extending SHERLOC.

3. SHERLOC PRINTOUT AND INTERNAL LOGIC.

SHERLOC consists of two modules: SECRIT and WATSON. SECRIT stands for SHERLOC's External Communication Routine, Initiator and Terminator. WATSON stands for Way of Accumulating and Taking Samples Obviously Needed.

SHERLOC
Something to Help Everybody
Reduce Load On Computers

SECRIT	WATSON
SHERLOC's External Communication Routine Initiator and Terminator	Way of Accumulating and Taking Samples Obviously Needed

FIGURE 1

A. SHERLOC Initialization. (Figure 2). When SHERLOC is started by the operator (Box A1), SECRIT receives control and requests information such as sampling rate from the operator (Box B1). SECRIT then attains supervisor state and key of zero via a B-transient. SECRIT modifies the external new program status work (PSW) and the supervisor call (SVC) new PSW to point to WATSON. SECRIT makes all of DOS timer-interruptable by turning on the external interrupt bit in operands of the supervisor's Set System Mask instructions and in all new PSW's except machine-check. SECRIT sets the Sample flag = don't-sample which WATSON will later use to determine whether to take a sample. SECRIT then sets the operator designated interval in the hardware timer (Box L1). SECRIT adjusts the DOS software clock so that the software clock minus hardware time will always give DOS the real time of day. SECRIT then WAITs on an Event Control Block (Box B2).

B. When the Timer Underflows. When the timer interval elapses, the hardware timer underflows (Box A3) causing an external interrupt. The external new PSW becomes the current PSW immediately upon timer underflow since SECRIT has modified DOS (Box G1) so that external interrupts are always enabled. WATSON receives control (Box B3) because SECRIT changed the address in the external new PSW during initialization (Box F1). WATSON then checks the Sample flag (Box B3) previously set by SECRIT (Box H1). Since this flag initially indicates samples are not to be taken, WATSON sets another interval in the hardware timer (Box G3), adjusts the software clock, and loads the external old PSW (Box H3). This returns control to whatever was occurring when the previous interval elapsed. This loop (Boxes A3, B3, G3, H3) in which the interval elapses but no sample is taken will continue until the external interrupt key is pressed (Box A5). When pressed, an external interrupt occurs, giving control to WATSON. WATSON inverts the Sample flag (Box B5) and then tests it (Box F5). At this time in our example, Sample flag = sample. Therefore, WATSON returns

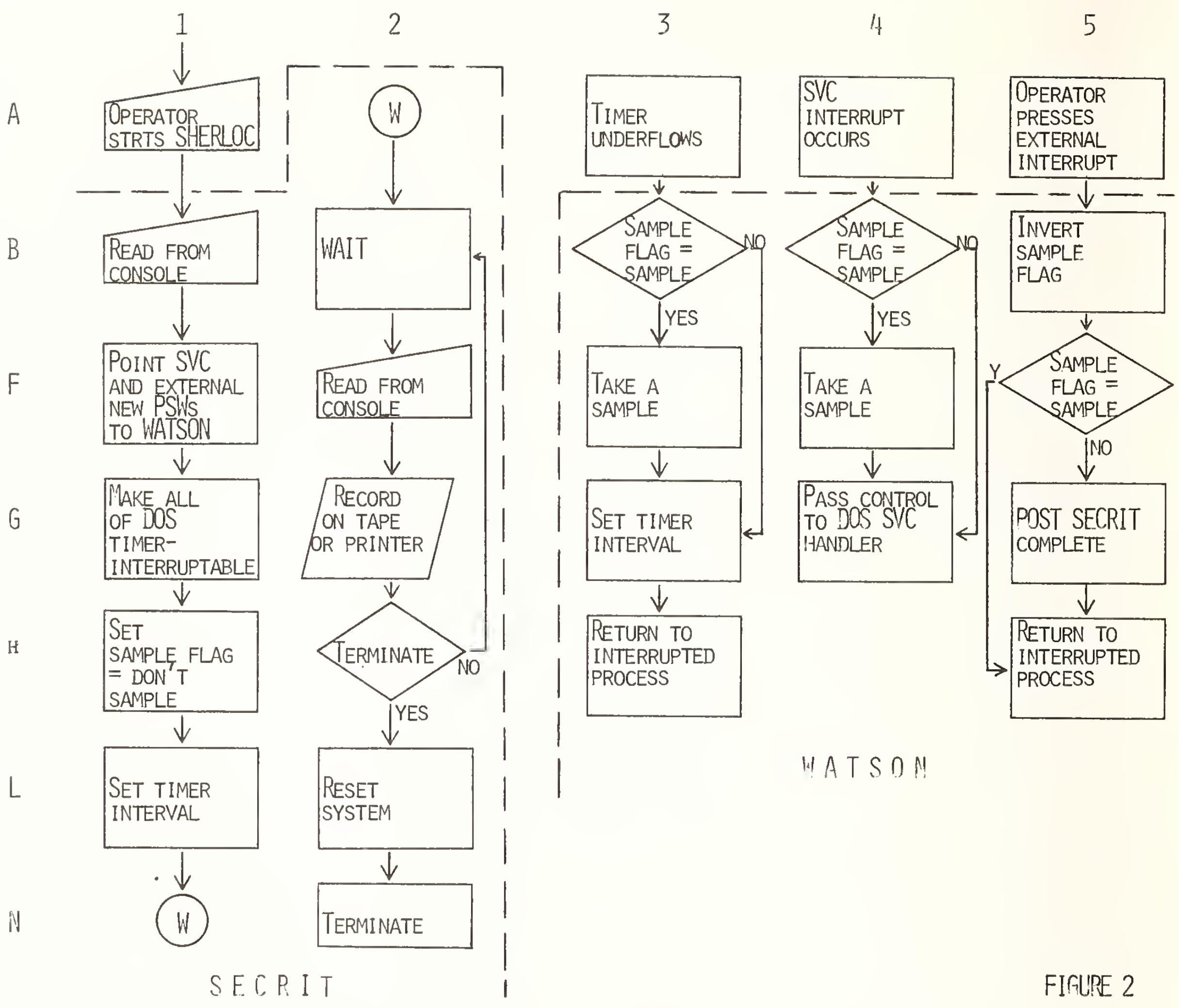


FIGURE 2

to the interrupted process (Box H5) by loading the external old PSW. The next time the timer internal elapses (Box A3), WATSON will test the Sample flag (Box B3) and decide to take a sample (Box F3).

C. Taking a Sample Due to Timer Underflow. A sample is taken by incrementing various buckets in core. (See Appendix 1). In the TOTAL TABLE, the SAMPLE INTERRUPTS and TOTAL INTERRUPTS buckets are incremented during every sampling. The CPU SAMPLE INTERRUPTS or WAIT SAMPLE INTERRUPTS bucket is incremented based on whether the external old PSW indicates the computer was in the CPU or WAIT state, respectively.

Regarding the TASK TABLE, there are seven tasks in DOS with no multi-tasking. Three are the partitions. The combination of the other four are generally thought of as the supervisor. Each task may be using the CPU, or may be explicitly (ie voluntarily) waiting for something other than the CPU (eg I/O), or may be implicitly (ie involuntarily) waiting for the CPU because some higher priority task is using the CPU. From the SVC 7 and SVC 2 bound bits in each task's Program Information Block (PIB), WATSON can determine which buckets to increment in the EXPLICIT and IMPLICIT WAIT columns of the TASK TABLE. If the external old PSW indicates the computer was in the CPU state, then the Program Interrupt Key (PIK) indicates which bucket in the CPU column to increment.

In the CORE TABLE, the first column gives the hexadecimal address of the starting byte of 16 byte intervals throughout core. When WATSON is taking a sample, it finds the resume PSW of each explicitly waiting partition in that partition's save area. It determines whether a partition is explicitly WAITing from the SVC 7 and SVC 2 bits in the PIB. WATSON then increments the WAIT SAMPLES bucket associated with the address in each resume PSW. The resume PSW address is the address of the instruction next to be executed. Furthermore, if the external old PSW indicates the computer was in the CPU state, WATSON increments the

CPU SAMPLES bucket associated with the address in the external old PSW. The remaining columns of the CORE TABLE are not created during sampling. At this point the sampling process (Box F3) is complete.

D. After Taking a Sample Due to Timer Underflow. WATSON then sets another interval in the hardware time (Box G3), adjusts the software clock, and loads the external old PSW to resume normal system processing (Box H3).

E. When an SVC Interrupt Occurs. SVC instructions are executed by both the supervisor and problem programs. Problem programs execute SVC's to ask the supervisor to initiate I/O, to fetch core image modules from disk into core, etc. When an SVC is executed, an SVC interrupt occurs (Box A4) and the SVC new PSW becomes the current PSW. WATSON then receives control since SECRIT changed the address in the SVC new PSW during initialization (Box F1). If the Sample flag indicates not to take a sample (Box B4), control is passed to the DOS SVC interrupt handler (Box G4) to handle the SVC. On the other hand, if a sample is to be taken (Box F4) (See Appendix 2), either the SUPERVISOR or PROBLEM PROGRAM bucket at the top of the FETCH TABLE is incremented based on the supervisor state bit in the SVC old PSW. If the SVC is an SVC 1, 2 or 4, the FETCH bucket is incremented, and the name pointed to by register 1 is either found in the table and its bucket incremented, or else added to the table with bucket equal to one. This completes the SVC sampling process (Box F4). Control is then passed to the DOS SVC interrupt handler (Box G4).

F. When the Operator Presses External Interrupt to Record Data. If the operator presses the external interrupt key (Box A5) when the Sample flag = sample, WATSON receives control, inverts the Sample flag to don't-sample, POSTs SECRIT complete (Box G5) and returns to the interrupted process. SECRIT then (Box F2) reads commands from the operator. If the operator indicates that the accumulated samples in core are to be recorded on tape and/or printer, SECRIT accomplishes the recording (Box G2).

(See Appendix 2) During printing, the last six columns of the CORE TABLE are calculated. The CPU SAMPLE PERCENT column is the CPU SAMPLE column divided by the SAMPLE INTERRUPTS in the TOTAL TABLE. The WAIT SAMPLE PERCENT column is the WAIT SAMPLES column divided by the SAMPLE INTERRUPTS. The TOTAL SAMPLE PERCENT column is the sum of the previous two columns. The next three columns are running totals of the corresponding previous three columns.

After recording on tape and/or printer (Figure 2, Box G2), if the operator has not indicated that SHERLOC should terminate, SECRIT again WAITs (Box B2). The operator may then press external interrupt (Box A5) to resume sampling. On the other hand, if the operator has indicated that SHERLOC should terminate (Box H2), SECRIT resets the previously modified new PSW's and Set System Mask operands to their original status, and goes to End-of-Job (Box N2).

4. USES TO DATE.

There have been primarily two uses of SHERLOC to date.

A. Supervisor Improvement. The Computer Systems Command had decided to change from its previous 8K supervisor to a 20K supervisor, primarily due to additional required and desired features. On a 360-30, it was found that the additional features in the 20K system caused a decrease in computer thruput of between 20% and 30% compared with the 8K system. SHERLOC was developed to improve this thruput.

(See Appendix 3). The supervisor options and other computer system characteristics for the 8K and 20K systems are shown in the columns labelled 1 and 8, respectively. The general plan was to compare, for each option, the degradation against the benefits gained. The largest unknown was the degradation of each option. SHERLOC was used to obtain a rough approximation of the degradation of each option. For supervisor options, the approximations were developed from matching, by absolute address, the supervisor source coding for each option with the CORE TABLE printout. This provided the CPU time spent in each option. This CPU time was in some cases only approximate since for some options the code is scattered in the supervisor. This CPU time was then multiplied by an educated estimate of the percentage of constraint of this CPU time on thruput. This converted CPU time into approximate run time.

In the case of the job accounting option, JA=YES, it was decided, with help from the CORE TABLE, that this option should be treated as two suboptions. The first, (STEP) in column 8, keeps track of the start and stop time of each step. The second (CPU, WAIT, OVHD), keeps track of CPU and WAIT time by partition, and overhead time. To implement the separation of these two suboptions, the full option would be generated and the (CPU, WAIT, OVHD) coding no-op'ed.

A non-supervisor option under consideration was on which library each module would reside and in what order their names would appear in the directory of a given library. This option affects search time. This is the next to last option in the figure. From DOS DSERV listings and the FETCH TABLE, it was concluded that the sort modules should be on the Private Core Image Library and that their names should be at the beginning of its directory. It was also concluded that no other library adjustments need be made. The improvement (or negative degradation) due to this change was easily approximated by calculation.

Now that an approximate run time degradation was known for each option, tests 2 thru 7 could be designed, ie, options could be grouped rationally, based on approximate degradation and feature desirability. An 'X' in columns 2 thru 7 indicates that the option was specified as in column 8 rather than as in column 1. The various supervisors and libraries were then created and run with live data. The results are designated as "Eustis" near the bottom of the figure. Test 1 took 3007 seconds.

The degradation for the other tests used 3007 as their denominator. None of the times in the figure include operator intervention. The options in test 4B were considered optimal. Tests were then run with an average volume of live data. The results are designated as "Meade" at the bottom of the figure. The resulting "Meade" 4B degradation was -0.7% instead of the original 20K supervisor degradation of between 20% and 30%. This improved supervisor currently runs at about 20 sites and is expected to soon run at 40. Assuming the following factors, the results of this study will save about $50 per hour * 20 hours per day * 300 days per year * 40 sites * 25% degradation reduction = $3,000,000 per year.

B. SPOOL Improvement. The second use of SHERLOC has been in speeding up a SPOOL program on 360-30's. Generally our user programs write print data to tape or disk and writes to a printer. It was discovered that a background process would run about 37% slower when SPOOL was running in foreground than when nothing was running in foreground. (See Appendix 1). By SHERLOCing this SPOOL running alone, the CORE TABLE revealed that supervisor CPU was 33.57% and problem program CPU was 1.26%. Each of these percentages was calculated by subtracting the CUMULATIVE CPU SAMPLE PERCENT at the beginning of the appropriate core area from that at the end. Furthermore, the CORE TABLE indicated that the large supervisor CPU usage was due to the unchained printer channel command words since the problem program was asking the supervisor to handle each print or space operation separately. By changing SPOOL to chain up to 20 print and space operations together, the supervisor would handle only one twentieth as many requests. In our test environment, this reduced the supervisor CPU to 3.98% and the background degradation to 9%. Since this SPOOL currently runs at about 20 sites and is expected to soon run at 40, this improvement will be substantial.

5. FUTURE SHERLOC IMPROVEMENTS.

We intend to change SHERLOC so that it can be run without usual operator assistance. This means that parameters will be entered by card rather than console, and that SHERLOC will start and stop sampling based not on pressing external interrupt, but perhaps on the starting and stopping of the job being monitored. We also intend to write each sample to tape rather than incrementing appropriate buckets. Later another program would read these samples from tape, increment buckets and print. This would reduce core required from the current 40K to about 4K. We would also complete the documenting of SHERLOC. At that time it could be used by others. Later, we may add capabilities such as monitoring the percentage of time each device and channel is busy.

6. CONCLUSION.

In conclusion, SHERLOC's major advantage over most, if not all, other software monitors is that it displays supervisor CPU time by core address. This is possible because WATSON operates above the operating system. SHERLOC has assisted in increasing thruput on twenty 360-30's and 40's by indicating modifications to our supervisor and libraries, and is currently aiding us in improving our SPOOL program. Developing SHERLOC took one man-month. Improving the supervisor and libraries took three man-months. These four man-months of effort are expected to save $3 million per year.

```
MONITOR V.3 CORE MAP/COUNTER REPORT...F2 JOB MON          OF  04 52 01 19/11/73

OLD SPOOL IN F1.  9891 RECORDS.  SPOOL TEST .  FT MYER.

SAMPLE INTS  CPU SAMPLE INTS  WAIT SAMPLE INTS  TOTAL INTS
0010893        0003795          0007098           0010893   <---- TOTAL TABLE

                 CPU      EX WAIT    IM WAIT
ALL BOUND      0001566    0000000    0009327
BG             0000000    0000000    0000000
F2             0000000    0010893    0000000 <---- TASK TABLE
F1             0002229    0008129    0000535
ATTN ROUTINE   0000000    0000000    0000000
QUIESCE        0000000    0000000    0000000
SUPERVISOR     0000000    0000000    0000000                    CORE TABLE
                                                                /
                                                      CUM      /CUM       CUM
                         CPU        WAIT     TOTAL    CPU      WAIT       TOTAL
       CPU      WAIT     SAMPLE     SAMPLE   SAMPLE   SAMPLE   SAMPLE     SAMPLE
ADDR   SAMPLES  SAMPLES  PERCENT    PERCENT  PERCENT  PERCENT  PERCENT    PERCENT

00150  00195             001.79 \            001.79   001.79              001.79
00160  00378             003.47 |            003.47   005.26              005.26
00170  00111             001.01 |            001.01   006.27              006.27
00180  00351             003.22 |            003.22   009.50              009.50
00190  00206             001.89 >SUPERVISOR  001.89   011.39              011.39
001A0  00035             000.32 |AREA:       000.32   011.71              011.71
001C0  00078             000.71 |33.57-0.00  000.71   012.43              012.43
001D0  00037             000.33 |=33.57      000.33   012.76              012.76
001E0  00102             000.93 |            000.93   013.70              013.70
  .               .             |     .                  .                   .
  .               .             |     .                  .                   .
  .               .             |     .                  .                   .
010A0  00002             000.01 |            000.01   031.17              031.17
01F50  00153             001.40 |            001.40   032.58              032.58
01F60  00004             000.03 |            000.03   032.61              032.61
01F70  00041             000.37 |            000.37   032.99              032.99
01F80  00063             000.57 /            000.57   033.57              033.57
123C0           10893              100.00    100.00   033.57   100.00     133.57
1D860  00001                    \                     033.58   100.00     133.58
1D870  00010             000.09 |            000.09   033.67   100.00     133.67
1D880  00011             000.10 |            000.10   033.77   100.00     133.77
1D890  00021             000.19 |            000.19   033.96   100.00     133.96
1D8A0  00002             000.01 |            000.01   033.98   100.00     133.98
1D8B0  00002             000.01 |            000.01   034.00   100.00     134.00
1D8C0  00010             000.09 |            000.09   034.09   100.00     134.09
1D8D0  00004             000.03 >PROBLEM     000.03   034.13   100.00     134.13
1D960  00002             000.01 |PROGRAM     000.01   034.15   100.00     134.15
1D970  00009             000.08 |AREA:       000.08   034.23   100.00     134.23
1DA30  00008             000.07 |34.83-33.57 000.07   034.30   100.00     134.30
1DA50  00037             000.33 |=1.26       000.33   034.64   100.00     134.64
1DA80  00011             000.10 |            000.10   034.74   100.00     134.74
1DEE0  00005             000.04 |            000.04   034.79   100.00     134.79
1DFA0  00003    08129    000.02 |  074.62    074.65   034.82   174.62     209.44
1DFB0  00001                    |                     034.82   174.62     209.45
1E320  00001                    /                     034.83   174.62     209.46
```

APPENDIX 1

FETCH TABLE

```
    SVC  FETCH  COUNTS                                                                =FETCH
000000000000085074  =SUPV        000000000000044623   =PPGM   000000000000006026

$$BATTNG     00001    $JOBCTLG     00108    $$GTPIKC     00036    $$BEOJ4

$$BEOJ       00106    $JOBCTLA     00105    $$BATTNU     00001    $JOBCTLJ

$$GTPIK      00035    $$BOPENR     00106    $$BOPNR2     00038    $$BOPEN

$$BOMSG2     00005    $$BOMSG5     00001    $$BEOJ2      00001    $$BPSW

$JOBCTLD     00019    $$BOCP01     00041    $$BCLOSE     00261    $JOBCTLK

$$BCBLIS     00027    $$BCBUSR     00133    $$BOPEN2     00098    $$BOMT01

$$BOMTES     00030    $$BOIS01     00044    $$BOIS02     00044    $$BOIS03

$$BOIS04     00044    $$BOIS05     00044    $$BOIS06     00044    $$BOIS07

$$BSETFL     00010    $$BSETFF     00010    $$BSETFG     00010    $$BOSD00

$$BOSIGN     00157    $$BOSDI2     00066    $$BOSDI4     00066    $$BOSDC1

$$BCMT01     00028    $$BENDFL     00012    $$BENDFF     00010    $$BCLOS2

$$BOUR01     00003    SORT         00016    SORTRCL      00016    SORTRCB

SORTRCD      00025    SORTRCC      00016    $$BOSD01     00091    $$BOSD02

SORTRCN      00016    SORTRSD      00016    SORTASA      00016    SORTROA

SORTRBA      00016    SORTRPB      00016    SORTRSM      00016    SORTROC

$$BOMTOM     00011    $$BOMT04     00024    LSF100R2     00001    $$BCBUSW

L12ATN19     00003    L12ATN13     00015    L12ATN17     00004    L12ATN18

LSF13200     00001    LSF134E0     00001    SORTRSN      00012    SORTRGB

L11ATN30     00001    LSF136E1     00001    LSF136R1     00001    LSF136R2

BAL10VRA     00002    BAL10VRB     00001    $$BSETL1     00157    BAL10VRC

LSF13830     00001    L11ATN36     00001    L12ATN15     00001    LSF13835

LSF13850     00001    LSF13860     00001    LSF13870     00001    LSF140B1

LSF14200     00001    LSF14300     00001    LSF14350     00001    LSF14400

L12ATN14     00001    LSF14800     00001    LSF160E1     00001    LSF16000
```

APPENDIX 2

	Existing 8K.	Functionally required or clearly advantageous.	Probably advantageous.	Universality.	Universality plus seek separation.	Universality plus direct access file protect.	Questionable.	Double peripherals: clearly undesirable.	CPU-wait-ovhd job accounting: clearly undesirable.	Original 20K V5.
TEST # =	1	2	3	4	4A	4B	5	6	7	8
MPS=	YES	X	X	X	X	X	X	X	X	BJF
TP=	NO			X	X	X	X	X	X	BTAM
EU=	NO	X	X	X	X	X	X	X	X	YES
ERRLOG=	NO						X	X	X	YES
IT=	NO		X	X	X	X	X	X	X	F2
TEB=	5	X	X	X	X	X	X	X	X	13
TEBV=	NO						X	X	X	(DASD,13)
SKSEP=	NO				X		X	X	X	16
CE=	NO		X	X	X	X	X	X	X	800
JA=	NO		X	X	X	X	X	X	X	[STEP]
JA=	NO						X	X	X	[S10]
JA=	NO								X	[CPU,WAIT,OVHD]
PTO=	NO		X	X	X	X	X	X	X	YES
PCIL=	NO	X	X	X	X	X	X	X	X	YES
CBF=	NO		X	X	X	X	X	X	X	5
CCHAIN=	NO			X	X	X	X	X	X	YES
AB=	NO		X	X	X	X	X	X	X	YES
WAITM=	NO		X	X	X	X	X	X	X	YES
DASDFP=	NO					X	X	X	X	(1,2,2314)
[PGM=	25,10,5	X	X	X	X	X	X	X	X	100,100,22]
JIB=	25	X	X	X	X	X	X	X	X	80
CHANQ=	15	X	X	X	X	X	X	X	X	40
IODEV=	17	X	X	X	X	X	X	X	X	40
[peripherals	single							X	X	double]
[SVC 51	NO		X	X	X	X	X	X	X	YES]
[assgn tapes to	2	X	X	X	X	X	X	X	X	1]
[sorts on PCIL	NO	YES	YES	YES	YES	YES	YES	YES	YES	NO]
[JCL, file alloc, libraries,workarea	8K	X	X	X	X	X	X	X	X	20K]
Eustis: Run time in seconds.	3007	2671	2701	2710	2748	2717	2905	2950	3101	3794
Eustis: Degradation in % =(T-3007)/3007.	0.0	-11.2	-10.2	-9.9	-8.6	-9.6	-3.4	-1.9	+3.1	+26.2
Meade: Run time in seconds.	8245	7987				8186				
Meade: Degradation in % =(T-8245)/8245.	0.0	-3.1				-0.7				

[ ] Logical, not actual, specifications.

APPENDIX

BENCHMARK EVALUATION OF OPERATING SYSTEM SOFTWARE: EXPERIMENTS ON IBM'S VS/2 SYSTEM

Bruce A. Ketchledge

Bell Telephone Laboratories

This paper concerns the problem of evaluating a new and unfamiliar operating system using benchmarking techniques. A methodology for conducting a general evaluation of such a system is presented, and illustrated with a recent study of IBM's new VS/2 system. This paper is broken into 2 major sections:

1. An Operating System Evaluation Methodology.

2. A Case Study: Evaluation of IBM's VS/2 Operating System.

In the first section, an operating system evaluation methodology developed in the Business Information Systems Support Division at Bell Laboratories is presented. The second section illustrates the application of this technique in a recently-completed VS evaluation project. Before plunging into the main parts of this paper, a little background (and a few caveats) are needed.

This paper will concern, specifically, methods of conducting a <u>general</u> evaluation of a <u>batch only</u> operating system. The use of the word 'general' implies that the type of evaluation project dealt with would not be aimed at predicting the performance of an already-coded application on the system (utilizing a particular central processor). Rather, the situation we envision is the case where the general performance-effecting features of the operating system are to be identified and probed to a sufficient depth to allow 'ball-park' estimates of the system's performance (e.g., overheads) in a given processing environment.

Moreover, this paper looks at batch-only systems, although much of the general methodology described might well apply in time-shared or batch/time-shared environments.

As this paper is the outgrowth of an evaluation project concerning a particular operating system (IBM's VS/2 Operating System), it might be well at this point to briefly sketch the historical background of this project. Shortly after the IBM announcement of virtual systems on August 2, 1972, the Business Information Systems Support Division at Bell Laboratories made a decision to do a general evaluation of these new systems. IBM's clear commitment to virtual systems, combined with the many new features of these systems, made the need for such an evaluation imperative.

Up until the time an IBM virtual system was available in-house (April 1973), the main part of the evaluation effort centered, necessarily, on reading IBM-supplied documentation concerning VS/1 and VS/2. As will be discussed in Section 2, once a 'live' VS system was at hand, an experimental/benchmark approach was used in gaining familiarity with the system.

Once familiarity with the system was gained, formal benchmark tests were conducted and the resulting data were analyzed. Seen in its entirety, it was felt that this evaluation project followed a pattern of sufficient generality and interest to be abstracted in the form of a general approach to operating system evaluations. It is this methodology we discuss in the next section.

1. <u>An Operating System Evaluation Methodology</u>

We propose to attack the problem of conducting a general evaluation of a batch operating system in four phases. The principal motivation behind formalizing the evaluation process and dividing it into four functional phases is to assure that no major characteristic of the system being examined nor any potentially useful approach to that system is excluded from consideration. As will be pointed out below, the particular approach proposed also has the benefit of facilitating management of the project in cases where more than one person is involved.

Some caveats to be borne in mind concerning the methodology to be proposed are the following:

- Vendor cooperation.
- Available system.
- Internal documentation.

Our approach depends heavily on the vendor's cooperation, and having available (perhaps through the vendor),a 'live' system to test as well as internal documentation of the operating system. Generally, the vendors have been cooperative, especially if the system to be tested is a new offering. In any event, the most crucial requirement for a successful evaluation effort is the availability of a 'hands-on' system. In the sequel we shall assume all of the above elements are present during the project.

The phases of the evaluation project are listed on the next page, along with the rough percentage of the total evaluation effort each might be expected to consume.

EVALUATION PROJECT PHASES

1. Analysis of Vendor Documentation/Establishment of Project Goals. (30%)
2. Development of Experiments/Tools. (20%)
3. Testing of Experiments/Tools. (20%)
4. Conduct formal benchmark experiments, analyze data. (30%)

The phases are named in such a way as to suggest (roughly) their basic functions. Note that the percentages shown represent only typical amounts of the total effort (in man-months) devoted to that phase. Also note that according to this table, 70% of the effort is concentrated in the pre-benchmark phases. . .the significance of this fact will be discussed below.

The first phase of the evaluation project involves analysis of vendor documentation and development of 'working hypotheses'. This is a crucial phase of the project (and should consume at least 30% of the total effort). During this phase the following activities occur:

- Analysis of vendor 'high level' (overview) documentation.
- Analysis of vendor 'low-level' (internals) documentation.
- Establishment of reasonable goals for the project, taking into consideration factors such as vendor support, complexity of the system to be modeled, benchmark time available, etc.
- Development of one or more 'working hypotheses' about the system.

Analysis of vendor overview documents is especially important if a 'human factors' evaluation of the system is desired (i.e., ease of use, complexity, debugging aids, etc.). Internal documentation must be analyzed with special care if potential sources for one or more software or hardware probes are to be recognized. For example, if the system keeps tables of important internal events or queues, it may be worthwhile to gather that data (with software) in order to aid in performance evaluation. This technique will be illustrated in Section 2.

Goals can be set for the project only when some familiarity with the operating system has been developed. Influencing the goals are factors including:

- Available resources (time, manpower, computer time).
- Complexity (novelty) of the operating system.

Once some familiarity with the system has been gained, and goals for the project set, it is important to structure the project around one or more central 'themes'. In many cases this central theme, or 'working-hypothesis', is suggested by an outstanding or novel feature of the system to be examined. In some cases a 'straw-man' may be set up, which then motivates experiments designed to knock it down. In any event, such working hypotheses about the system serve to keep the experiments proposed in Phase 2 of the effort relevant to the goals set in Phase 1. Section 2 of this paper offers an example in which a comparison with the vendor's other major operating system provided an all-important working-hypothesis.

Phase 2 of the project involves capitalizing on the knowledge gained in Phase 1 by developing specific experiments and tools designed to probe important areas of the operating system. Some, though not all, of the experiments may be aimed at challenging the working hypothesis developed in Phase 1. Other experiments may be aimed at directly comparing throughput of the new operating system of the same vendor. Generally, each experiment will be handled by one individual (including its design, testing, and benchmark runs) and will include 3 elements:

- Software to stimulate the system ('Job mix').
- Measurements tools - hardware monitor, software monitor, log files, etc. This category includes software to reduce the collected data.
- Detailed specification of the experiment.

The evaluation team may decide to allocate one member of the team to handle some functions common to all the experiments (ie, hardware monitoring). The detailed specification of each experiment, however must be done on an individual basis, especially if one goal of the benchmark is the development of a real familiarity and self-confidence about the system by members of the team (these experimental designs must be reviewed by the team leader, of course). During the second Phase new tools needed to probe sensitive areas of the system may have to be developed. The tools may serve either or both of the validation and measurement functions.

The validation function concerns the checking of important system routines for operation that is consistant with the system's logic (as documented by the vendor) and consistent with system initialization parameters. For instance, in a VS/2 system initialization ('IPL') parameters may be set so as to effectively prevent the VS/2 Dispatcher from 'deactivating' (marking non-dispatchable) user tasks due to excessive paging. During the VS/2 evaluation project discussed in this paper, the systems parameters were set so as to 'turn-off' task deactivation, and the effect of these parameter settings was checked independently by a software monitor which examined the systems' Nucleus region and validated that no user tasks were being deactivated. (The use of this monitor is further illustrated in Section 2 of this paper). Particularly with a new operating system release, 'bugs' in the system may cause entire benchmark runs to be invalidated without causing the system to 'crash' (thus signalling the difficulty). Passive protection can be provided by a software monitor or trace facility, such as described above, which checks for normal operation of the system internals, especially immediately after system initialization. The measurements function, of course, is central to the entire effort. Software monitor techniques, in particular, are useful for measuring logical queues in the operating system as well as determining efficiency and stability of operating system resource-allocation algorithms. The integration of these measurements with more conventional measurement techniques (hardware monitors and log files) is illustrated in Section 2.

Phase 3 of the evaluation project involves testing and debugging of each experiment and tool on the system to be benchmarked. This is an excellent opportunity for each member of the team to acquaint himself with the system and gain a real feel for the response of the system to various stimuli (job streams). Ideally, this phase of the evaluation project should extend over a considerable period of time, to allow intuition about the system to grow as well as allow revamping of tools or experiments which prove invalid. This period also allows the sensitivity of the system to initialization parameters (in IBM jargon, 'IPL' parameters) to be determined (especially if nominal parameter values are to be selected for the venchmark itself).

Phase 4 of the project involves carrying out the formal experiments developed in the earlier phases, analyzing the resulting data, and documenting the results. In most cases, by the time this phase has been reached, most (70% or more) of the work has been done and the experimental designs crystalized. Experience has shown that relatively little can be done to correct or revamp experiments yielding invalid or unreproducible results in this phase... this 'debugging' must be done in Phase 3.

Certainly the most interesting part of the project is the data analysis which follows the benchmark runs. The working hypothesis must be refuted or validated at this point, and alternative explanations (based on internals knowledge gained in Phase 1) of system behavior advanced if necessary. It is crucial that all the experimental results be understood in terms of the internal structure of the operating system in order to be able to say the project was a success. Documentation of the results should include those explanations, for they may help others unfamiliar with the system comprehend and appreciate the benchmark results. Conclusions and recommendations about the operating system should be written up as a separate section of the final report, and should consider the experimental results obtained in their totality, as well as any other knowledge or intuition gained about the system during Phase 3.

While many of the points made above were very general, it is easy to give specific examples of their implementation in the context of the recently-completed evaluation of VS/2 done at Bell Laboratories. In the following section the course of this evaluation project is followed in light of the discussion of Section 1.

2. A Case Study: Evaluation of IBM's VS/2 Operating System

Phase 1 (analysis of documentation) of the VS/2 project began with collection of VS/2 documentation. Primarily internals documentation of the operating system was examined, especially prominent in this role being the IBM-supplied 'Features Supplements' for VS/1 and VS/2. The hardware changes IBM introduced with the VS announcement also were examined, primarily through the 'Guide to the IBM 168' manual.

After the necessary background in VS was obtained, goals for the evaluation effort were set. It is worthwhile at this point to discuss in some detail the considerations which led to the goals established for the study, and to list the goals themselves.

The VS benchmark had several goals. Various factors inherent in the environment in which the tests were run conditioned these goals and restricted the type of tests which could be run. In particular, the hardware configuration used was a 370/145 CPU with 512K bytes of memory. The restricted amount of main storage available made it impossible to do a worthwhile comparison of OS and VS throughput. This was because OS/MVT 21.6 was found to occupy over half of main memory when running, thus restricting the memory available to user programs in such a way as to limit the number of user programs running simultanelusly to one or two at most. Thus any OS results obtained would be mostly a function of the lack of multiprogramming depth in the runs. The only alternative would have been to run jobs of unrepresentatively small memory (Region) requirements so as to allow more jobs to be active simultaneously under OS. This alternative was discarded.

Another conditioning factor for both the experiments and the resulting conclusions was the size, speed, and architecture of the 370/145 CPU. The 370/145 does not have the highly centralized control register structure, pipelining features, outboard channels, or cache memory of the larger 370 CPU's. Thus questions relating, for example, to the effect of a triple-hierarchy memory (cache, main memory, and virtual memory) on system performance could not be explored because of the 145's lack of a cache memory. Hardware monitoring was also more difficult on the 145 than on more 'centralized' CPU's like the 155 or 165. This fact, however, did not significantly hinder the study.

With the above factors in mind, and with an eye on Bell Laboratories' particular interests as a consultant to Operating Telephone Companies, the following goals were set for the benchmark:

1. Develop a better understanding of VS internals and spread this knowledge among others in Bell Laboratories as a whole.
2. Develop an 'intuitive feel' for the general performance of the 145 running under a VS operating system.
3. Lay the foundation for further benchmark and simulation efforts directed at VS through the development of tools and techniques for VS analysis.
4. Develop a data base of experimental results from a VS system that would be useful in validating future simultation and analytic models of VS.
5. Highlight areas of strength and weakness of VS to help Operating Telephone Companies (OTC's)

make informed decisions concerning acquisition of VS systems.
6. Shed light on possible areas of further exploration by Bell Laboratories personnel concerning VS performance factors.

The above goals were, as discussed earlier in this section, conditioned by the hardware restrictions of the system tested. In particular no comparison of OS and VS throughput was attempted. Rather the above goals emphasized in-depth probing of VS in its own right (goals 1 and 2), as well as the building of a foundation for a multi-faceted approach to the predictive question itself (goals 3 and 4). Goal 5 was included to meet the immediate needs of OTC data processing managers who must make dollars and cents decisions, based on such studies, concerning selection of hardware and software. Finally, any good study results in more new questions being asked than old questions being answered, and goal 6 reflected this fact. In total, these goals represent a rather typical example of goals for a general batch operating system evaluation project.

After these goals were set, the development of a working-hypothesis about VS/2 was facilitated by comparison with OS/MVT (on which VS/2 is based). Clearly, the main difference between these systems is in the area of memory management considering VS/2's <u>virtual memory</u> capability. Hence the working hypothesis arrived at was a rather simple one, i.e., that due to the overheads inherent in managing virtual memory the <u>paging</u> rates associated with the system would be the main determinant of system overhead and would have a great effect on overall throughput. Implicit in this hypothesis was the assumption that paging would be rather costly (in CPU cycles) in VS/2 and hence the system would degrade as soon as any significant amount of paging occured. This hypothesis was shown later to be false, but in the process of designing experiments that tested the hypothesis, much was learned about the system.

Phase 2 of the project was dominated by development of tools and experiments to test the working hypothesis as well as answer other questions about the VS/2 system. One highlight of this phase was the development of a software monitor, appropriately named VSMON, which was used both to validate the VS/2 operating system and measure its performance. The development of VSMON began in the first phase of the project, when specific tables used by the VS/2 Paging Supervisor to record the status of each 4K-byte <u>page</u> of main storage were discovered. These tables (<u>Page Vector Table and Page Frame Table</u>) also record the status of various logical queues in the VS/2 operating system. Hence it was decided that in order to validate and measure VS/2 Paging Supervisor performance a software monitor which would periodically access those tables would be built. The result was a tool useful in both measuring the ownership of real memory by user and system programs and in checking for correct operation of the Paging Supervisor. It is worth noting that in one instance, a potentially harmful 'bug' in VS/2 was found using the VSMON monitor; the affected benchmark experiment was rerun after the 'bug' was fixed. The measurements from the monitor have made possible (in combination with log file data and hardware monitor information) a much more complete picture of the VS/2 system than heretofore had been obtained.

Phase 3 of the project was spread over almost two months of time, during which each of the 5 major experiments was tested, and the VSMON monitor checked out. As a result of this extended 'dress-rehersal', the last phase of the benchmark came off without incident, each experiment operating as expected. As of the time of this writing, documentation of the project is in progress.

We believe that the methodology set forth in Section 1., and the case study presented in Section 2., provide a useful set of guidelines for those in the field who are anticipating general operating system evaluations.

REPORT ON FIPS TASK GROUP 13 WORKLOAD DEFINITION AND BENCHMARKING

David W. Lambert

The MITRE Corporation

ABSTRACT

Benchmark testing, or benchmarking, one of several methods for measuring the performance of computer systems, is the method used in the selection of computer systems and services by the Federal Government. However, present benchmarking techniques not only have a number of known technical deficiencies, but they also represent a significant expense to both the Federal Government and the computer manufacturers involved. Federal Information Processing Standards Task Group 13 has been established to provide a forum and central information exchange on benchmark programs, data methodology, and problems. The program of work and preliminary findings of Task Group 13 are presented in this paper. The issue of application programs versus synthetic programs within the selection environment is discussed. Significant technical problem areas requiring continuing research and experimentation are identified.

INTRODUCTION

Earlier this year the National Bureau of Standards approved the formation of FIPS Task Group 13 to serve as an interagency forum and central information exchange on benchmark programs, data, methodology, and problems. The principal focus of Task Group 13 is to be on procedures and techniques to increase the technical validity and reduce the cost of benchmarking as used in the selection of computer systems and computer services by the Federal Government.

In May, invitations were issued to Federal Government Agencies, through the interagency Committee on Automatic Data Processing and the FIPS Coordinating and Advisory Committee, for nominees for membership to Task Group 13. Invitations for industry participation were also issued through CBEMA. In response to these invitations we have received 26 nominees from various Federal agencies and 6 from industry. These names are being submitted to the Assistant Secretary of Commerce for approval and we hope to be able to start holding meetings in late January or early February of next year.

This morning I would like to cover the five main topics shown in the first vugraph (Figure 1). First, a brief review of the approved program of work for Task Group 13. Second, a review of some of the techniques and typical problems involved in the current selection procedures. Next, I will discuss some alternatives to the present methods used to develop benchmark programs and attempt to clarify a major issue in the selection community at this time: application programs or synthetic programs for benchmarking? As a fourth topic, I would like to review the general methodology of synthetic program development and some specific tools and techniques currently being used. I would like to conclude with an overview of some key technical problems in workload definition and benchmarking which require further research and experimentation.

FIGURE 1

REPORT ON FEDERAL INFORMATION PROCESSING STANDARDS TASK GROUP 13

WORKLOAD DEFINITION AND BENCHMARKING

- PROGRAM OF WORK
- REVIEW OF THE CURRENT SELECTION PROCESS
- BENCHMARK PROGRAMS: APPLICATION OR SYNTHETIC?
- THE METHODOLOGY FOR SYNTHETIC PROGRAM DEVELOPMENT
- SUMMARY OF PROBLEMS REQUIRING FURTHER STUDY AND EXPERIMENTATION

PROGRAM OF WORK

The presently approved program of work is summarized in the next vugraph (Figure 2). The first three tasks should be self-explanatory. Task 4 is probably the most controversial at this time because it is not really clear what type of benchmark programs should be included in any sharing mechanism such as a library. I will discuss this issue later in my talk. As part of this task we plan to test and evaluate a number of available benchmark programs. If you have programs and/or a test facility and would like to participate in such an activity, we would be pleased to hear from you. In Task 5, initially we plan to capitalize upon the experiences of people who have been involved in previous selections and to prepare a preliminary set of guidelines to be made available to users and others faced with selection problems. Federal Guidelines and Standards will possibly come later. The bibliography (Task 6) is intended to include articles and reports covering all aspects of workload characterization, workload generation, benchmarking, comparative evaluation, etc. — but limited to the selection environment. This task is well under way with more than 75 articles having been identified at this time.

FIGURE 2

PROGRAM OF WORK

1. FUNCTION AS A FORUM AND CENTRAL INFORMATION EXCHANGE
2. REVIEW CURRENTLY USED SELECTION PROCEDURES AND TECHNIQUES
3. IDENTIFY AND EVALUATE NEW TECHNICAL APPROACHES
4. DEVELOP AND RECOMMEND A MECHANISM TO FACILITATE SHARING OF BENCHMARK PROGRAMS
5. PREPARE FEDERAL GUIDELINES AND STANDARDS
6. DEVELOP A SELECTED BIBLIOGRAPHY

THE SELECTION PROCESS

For the benefit of those of you not familiar with the selection process, I have listed the typical steps used in most competitive procurements today in the next vugraph (Figure 3).

FIGURE 3

STEPS IN CURRENT SELECTION PROCESS

1. USER DETERMINES TOTAL WORKLOAD REQUIREMENTS FOR PROJECTED LIFE CYCLE OF SYSTEM.
2. USER SELECTS TEST WORKLOAD, TYPICALLY WITH ASSISTANCE OF APPROPRIATE SELECTION AGENCY.
3. SELECTION AGENCY PREPARES AND VALIDATES BENCHMARK PROGRAMS, DATA AND PERFORMANCE REQUIREMENTS FOR RFP.
4. VENDOR MAKES NECESSARY MODIFICATIONS TO BENCHMARK PROGRAMS AND RUNS ON PROPOSED EQUIPMENT CONFIGURATIONS.
5. VENDOR RUNS BENCHMARK TEST FOR USER AND SELECTION AGENCY FOR PROPOSAL VALIDATION PURPOSES.
6. VENDOR RERUNS BENCHMARK TESTS AS PART OF POST-INSTALLATION ACCEPTANCE TESTING.

Step 1 is typically only one aspect of the requirements analysis and preliminary system design phase for any new system. For upgrades or replacements to an existing system, there are a variety of hardware and software monitors and other software aids that can be used to collect data on the existing system. Various programs and statistical analysis techniques can be used to analyze this data to identify the dominant characteristics of the present workload. For example, a number of workers are beginning to use a class of cluster algorithms to aid in this step. The major problems occur for conceptual systems or systems in which the workload is expected to change radically in the future: for example, when changing from a batch environment to a time-sharing environment. In these cases, it is extremely difficult to estimate the future workload, particularly for the later years in the projected life cycle of the system.

In step 2, the general goal is to determine peak load periods or to compress long-term workloads — such as a 30-day workload — into a representative test workload that can be run in a reasonable length of time — say two hours. For batch, multi-programmed systems, the test workload is generally designed to have the same mix of jobs (and in some cases, sequence of jobs) as the total workload determined in step 1. For on-line systems, the test workload must also be designed to represent the mix of terminal jobs in the total workload. For an upgrade or replacement of an existing system, the peak period or representative mix can generally be identified or selected using resource utilization data. The test workload can also be verified by comparing its resource utilization data against the original data. For conceptual systems, of course, none of this experimental activity can take place.

Within the selection environment, the principal goal in step 3 is to develop machine-independent benchmark programs. In current selection procedures, the benchmark programs are generally drawn from among application programs obtained from the user's installation. This has a number of drawbacks due to differences in job control techniques, program languages, and data base structures from machine to machine. Also, in many cases the original programs were tailored to take advantage of some of the particular features of the user's machine. Most of these problems have been satisfactorily solved for batch, multiprogrammed systems. However, there are a host of new problems foreseen with the advent of general purpose data management systems, time-sharing and on-line transaction processing applications.

In step 4, the principal objective is to collect timing data for each configuration (original plus augmentations to meet a growth in workload) to be proposed in response to the RFP. Most problems and costs in this step are attributed to trying to get the benchmark programs to run on the vendor's own machine because of language differences and data base structure differences. In my own experience, seemingly simple problems such as character codes and dimension statements have been the cause for much aggravation in trying to get one program to run on another machine.

In steps 4, 5 and 6, there are major costs on the part of the vendors to maintain the benchmark facility, including equipment and personnel. There can also be significant scheduling problems if the vendor is involved in a number of procurements at the same time. These costs, and in fact all the vendor costs in preparing his proposals and running benchmarks, eventually get reflected in the costs of the computers to the Federal Government.

BENCHMARK PROGRAM DEVELOPMENT

Most of the technical problems and costs in the current selection process are generally attributed to having to select, prepare, and run a new set of programs for each new procurement. What are some of the alternative approaches that could be addressed by Task Group 13? The first one is probably the most obvious, and that is to develop tools and techniques to simplify the process of sanitizing a given user program and then translating it to other machines. This would include preprocessors to flag or convert machine-dependent job control cards, dimension statements, etc. I am not aware of any serious attempts or proposed attempts along this line; however, it seems to have some merit for consideration by Task Group 13.

A second approach, which has been proposed but never implemented even on an experimental basis, is to develop and maintain a library of application benchmark programs. These programs would then only have to be translated to other machines once. For each new procurement, the user would select some mix of benchmarks from this library which best approximates his desired test workload. The library would probably have to be extensive, since it would have to contain programs for a great variety of engineering, scientific and business applications. One proposal that I am familiar with specified 20 Fortran and 20 Cobol programs, all to have been parameterized to some extent to permit tailoring to a specific user's test workload. This approach seems to have been set aside by the selection community for a variety of reasons, the primary one probably being the cost to develop and maintain such a library. There is also some question as to the acceptance of this approach by the users. I believe that if the costs and benefits were analyzed on a Government-wide basis, this approach may be more favorable than it has appeared in the past.

The third approach, and the one receiving the most attention these days in the selection community, is the synthetic program, or synthetic job, approach. As most of you know, a synthetic program is a highly parameterized program which is designed to represent either (1) a real program or (2) the load on a system from a real program. The first form, which is a task-oriented program, is generally designed so that it can be adjusted to place functional demands on the system to be tested, such as compile, edit, sort, update, and calculate. The second form is a resource-oriented program which can be adjusted to use precisely specified amounts of computer resources such as CPU processing time and I/O transfers. Neither does any "useful" processing from an operational standpoint.

Typically, the test workload can be specified in terms of say 10 and 15 different types of task-oriented synthetics which, incidentally, may contain common parameterized modules such as tape read, compute, or print. In the resource-oriented synthetics, there is normally one original program which is duplicated and adjusted to represent individual programs in the test workload.

The motivation toward synthetics in the selection environment is of course the same as for an application benchmark library. The benchmark program or programs need only be translated and checked out once by each computer manufacturer, thus minimizing the costs involved in future procurements. The most serious technical question regarding synthetics seems to be their transferability from machine to machine, particularly the resource-oriented synthetics, since demands on resources are obviously dependent on the architecture of each machine. The task-oriented synthetics seem to offer the most promise within the selection environment primarily because they seem least system dependent. Also, user acceptance would be more likely if some means can be developed to express or map user-oriented programs in terms of task-oriented synthetic programs.

SYNTHETIC PROGRAM METHODOLOGY

The general methodology of the synthetic job approach is summarized in the next vugraph (Figure 4).

FIGURE 4
ELEMENTS OF SYNTHETIC JOB APPRDACH

1. COLLECT JDB AND SYSTEM DATA ON ACTUAL SYSTEM
2. SELECT SYNTHETIC PROGRAM TASKS DR MDDULES
3. CALIBRATE SYNTHETIC WORKLDAD
   - SET INDIVIDUAL PRDGRAM PARAMETERS
   - SET JDB MIX AND/OR JDB TIMING PARAMETERS
   - RUN SYNTHETIC WDRKLDAD DN TEST SYSTEM
   - COLLECT AND COMPARE DATA
   - REPEAT TO ACHIEVE DESIRED DEGREE OF CDRRELATION

Accounting programs are commonly used for data collection, although there is increasing use of hardware monitors, trace programs and other software packages to get more detailed timing data on the real (and the synthetic) jobs in execution. As has been pointed out earlier in this meeting, it is often necessary to use special techniques or develop special programs to be able to make specific measurements for the problem at hand. For example, in an experimental program sponsored by the Air Force Directorate of ADPE Selection, we are using a Fortran frequency analysis program to obtain run time data on actual programs. This data is in turn used to set the parameters of a synthetic Fortran program, while accounting data (SMF on an IBM 370/155) is used to compare the synthetic job to the original user job. For TSO jobs, we use a system program called TS TRACE to collect terminal session data, use this data in developing a synthetic session, and compare the two sessions on the basis of SMF data.

SUMMARY OF TECHNICAL PROBLEMS

In this last vugraph (Figure 5), I have listed some of the major problems affecting computer performance measurement and evaluation in general and computer selection in particular.

The lack of common terminology and measures affects all areas of selection, including workload characterization, performance specification, and data collection and analysis. The semantics problems alone make communication difficult between users and selection agencies and selection agencies and vendors.

FIGURE 5

SUMMARY OF PROBLEMS

- STANOARO TERMS AND MEASURES ACROSS MACHINES
- WORKLOAD OETERMINATION AND REPRESENTATION FOR CONCEPTUAL SYSTEMS
- TRANSFERABILITY OF SYNTHETIC PROGRAMS
- WORKLOAO SPECIFICATION AND GENERATION FOR ON-LINE SYSTEMS
- EFFECTS OF NEW COMPUTER SYSTEM ARCHITECTURES ON PRESENT METHODS

The next two problem areas have been discussed earlier. The fourth problem area is essentially how to benchmark on-line systems, particularly those with large numbers of terminals. There are a number of techniques in use at this time: operators at terminals following a given script; special processors to generate the terminal load; and special programs to process a magnetic tape with pre-stored message traffic. However, because of the lack of common terminal languages, it is necessary to develop a different script for each vendor's computer, which is not only costly but raises questions of equivalency across machines. For external drivers, such as the Remote-Terminal Emulator being developed for the Air Force Directorate of ADPE Selection, there arc also major implementation problems due to the large variety of terminals and the lack of standard data communications control procedures.

On the last point (in Figure 5) there is a new set of problems as a result of the trend toward virtual memory machines, computer networks and other computer-communications systems. In virtual memory systems, for example, synthetic benchmark programs have to be carefully constructed to accurately represent the dynamic addressing characteristics of user programs. There are practical problems of how to measure their addressing characteristics, as well as how to represent them in a synthetic program. Most synthetics currently under investigation for virtual memory machines contain matrix calculations which have the property that page fault rates can be adjusted (by changing the location of the data) while keeping the CPU execution times constant.

Problems such as these cannot be neglected by those involved in the selection process. It is hoped that Task Group 13 can serve an effective role in their clarification and in identifying and evaluating possible solutions.

PERFORMANCE MEASUREMENT AT USACSC

Richard Castle

U.S. Army Computer Systems Command

INTRODUCTION

The United States Army Computer Systems Command (CSC) was activated 31 March 1969 under the direction of the Assistant Vice Chief of Staff of the Army (AVCSA) and charged with the responsibility for the design, development, programming, installation, maintenance, and improvement of Army multicommand automatic data processing (ADP) information systems. These multicommand automatic data processing information systems must satisfy the information management requirements for as many as 41 identical data processing installations (DPI) located throughout the Continental United States, the Pacific and Europe. These systems include both management data systems and tactical data systems. The hardware configuration of the DPI's, on which the management information systems are executed, are not controlled by CSC, but rather are selected by the Army Computer Systems Support and Evaluation Command under the direction of the AVCSA and in accordance with cost effectiveness studies to determine the optimum hardware distribution for multicommand systems. Therefore, since hardware configuration changes would involve costly and time consuming optimization studies and acquisition procedures, the continually growing Army data processing needs requires that the throughput of the multicommand DPI's be maximized without reconfiguration.

This obviously can only be accomplished by reducing the run time of existing data processing systems and insuring that new systems, currently under development or soon to be fielded, are optimized for minimum run time. Thus, the need exists in the Computer Systems Command for program performance measurement, to reduce system program run time, and for configuration performance measurement, to identify hardware/software bottlenecks, caused by program changes or added workload from newly fielded systems.

APPROACH TO PERFORMANCE MEASUREMENT

The charter of the Computer Systems Command contains a requirement to conduct ADP research and development in order to keep abreast of the current state-of-the-art and to insure that the Command and its customers benefitted from technology advances. Recognizing the importance of performance measurement, the Commanding General assigned the responsibility for a research task to the Quality Assurance Directorate to develop a Command combinational hardware/software performance monitor. This task, R&D Task IV, Computer System Performance Monitor, was organized to accomplish the following milestones:

o Acquire and use existing monitors to evaluate the state-of-the-art.

o Evaluate the experience gained in using the existing monitors.

o Based upon this experience, develop specifications for a Command combinational hardware/software monitor to be used on multicommand ADP systems.

o Develop the monitor and place it in operation.

Task IV progressed to the point at which a hardware monitor was purchased, the X-RAY Model 160; several software monitors were leased, DPPE, DCUE, CUE and Johnson Job Accounting; the monitors were used in operational environments; and specification development began from the experiences gained in the use of the monitors. At this point, it became evident that industry was keeping abreast of and in some instances leading our specification development in their monitor development programs. Indeed, it appeared that by the time milestone 4 was reached, the desired monitor would already be available off-the-shelf from industry.

Consequently, the decision was recently made to attempt to leap-frog industry and start development of automated evaluation of performance monitor data. This research task was assigned to the Advanced Technology Directorate and the performance monitors currently in the inventory were to be put to more intensive operational use by the Quality Assurance Directorate.

APPLICATION OF PERFORMANCE MEASUREMENT AT CSC

The inventory of performance monitors currently being used at USACSC are shown in Figure 1.

These performance monitors are used by CSC throughout the multicommand ADP systems life cycle as shown in Figure 2. Figure 2 also shows the application of performance monitors by the Quality Assurance Directorate during the life cycle.

Early in the programming and documentation phase, as soon as object programs are available, program performance measurement can begin and program run time minimized. In the later phases of system integration and prototype testing both problem program and system performance monitors are used to aid in integration and reduce run time. Later in the life cycle, during system installation and after the system is operational, performance measurements and analyses are made to insure that no bottlenecks have crept into the system.

MONITOR	TYPE	VENDOR	OWNED/ LEASED	TRAVEL LIMIT	CAPABILITIES
X-RAY 160	Hardware	Tesdata	Owned	None	32 Counter/Timers 2 Distributors 96 Probes
DPPE	Software	Boole & Babbage	Owned	Multicommand Sites	IBM 360 DOS Problem Programs
DCUE	Software	Boole & Babbage	Leased	1 Site Per Lease	IBM 360 DOS Systems
Johnson Job Accounting	Software	Johnson Sys, Inc.	Leased	1 Site Per Lease	IBM 360 OS or DOS Systems
LEAP	Software	Lambda Corp.	To Be Leased	None	IBM 360 OS Programs Systems & Accounting

FIGURE 1 Quality Assurance Monitor Inventory

PHASE	PLANNING & DEFINITION	LIFE CYCLE DEVELOPMENT AND INSTALLATION				OPERATION & MAINTENANCE
		SYSTEM DESIGN	DETAILED DESIGN	PROGRAMMING AND DOCUMENTATION	TEST AND INTEGRATION	
EVENTS	FINAL RQMTS REVIEW	INITIAL DESIGN REVIEW	PDR CDR	DEBUG TESTING	PROGRAM TESTING INTEGR TESTING PROTO TEST	OPERATIONAL SYSTEM TEST MAINT & MOD
PRODUCTS	● GFSR ● ASRA DFSR ● SYSTEM/ PROGRAM SPECS ● SYSTEM ANALYSES	● DESIGN ANALYSES ● PROJECT PLANNING ● PROGRAM/MODULE NARRATIVES/FLOWS ● TEST CRITERIA		● PROGRAM NOTEBOOKS ● TEST PLANS & PROCEDURES	● TEST ● TEST REPORTS ● PERFORMANCE ANALYSIS	● OPERATIONAL PERFORMANCE ANALYSIS ● OPERATIONAL REVIEWS ● CHANGE ANALYSIS

FIGURE 2 Life Cycle

These performance monitors are used by CSC throughout the multicommand ADP systems life cycle as shown in Figure 2. Figure 2 also shows the application of performance monitors by the Quality Assurance Directorate during the life cycle.

Early in the programming and documentation phase, as soon as object programs are available, program performance measurement can begin and program run time minimized. In the later phases of system integration and prototype testing both problem program and system performance monitors are used to aid in integration and reduce run time. Later in the life cycle, during system installation and after the system is operational, performance measurements and analyses are made to insure that no bottlenecks have crept into the system.

SOME APPLICATION EXAMPLES

The monitors available to CSC have been used for several different applications in the past two years. Some of the applications that were important for their initial instructiveness in the use of monitors were the Base Operations System (BASOPS) II Timing Study, the Theater Army Support Command (Supply) System (TASCOM)(S) monitoring effort and the Standard Installation and Division Personnel System (SIDPERS) prototype tests. The Standard Army Intermediate Level (Supply) System (SAILS) Run-Time Improvement Study is the most recent and most successful application of performance measurement and therefore, is presented in the greatest detail here.

BASOPS II Timing Study.

The purpose of the BASOPS II project was to incorporate expanded functional capabilities into the SAILS program to incorporate the operational Standard Finance (STANFIN) programs and to include a newly developed SIDPERS package. The purpose of the timing study was to predict if the three subsystems would operate in a targeted 16 hour time frame on the standard BASOPS S360/30 configuration with core extended to 128K bytes. The Boole & Babbage DPPE software monitor was used to form the basis for the computations of the timing study. Completed SAILS programs were monitored on an IBM S360/30 times. Based on these figures the remaining program run times were computed and it was determined that the proposed system daily cycles would require in excess of 20 hours at five of the largest BASOPS installations. Nine medium size installations would require more than 15, but less than 20 hours. The remaining installations were estimated at less than 15 hours run time. As a result of this study, the SAILS and SIDPERS subsystems are currently undergoing run time reduction studies.

TASCOM(S).

The Theater Army Support Command (Supply), TASCOM(S), supports the US Army Material Management Agency, Europe (USAMMAE), which is a theater inventory control center having centralized accountability for command stocks in Europe. The system utilizes two IBM S360/50 I mainframes with 32 IBM 2401 tape drives and 7 shared IBM 2314 disk units. The operating system is MFT II and is supported on line with five IBM 1403 printers, one IBM 2501 card reader, two IBM 2540 card reader punches and sixteen IBM 2260 remote display stations for on-line inquiry to the system. The Boole & Babbage CUE Software monitor was used to monitor the configuration utilization. The system was monitored for forty-nine and one-half hours and it was found that the channel usage was unbalanced and the number of loads from the SVC-Library were excessive. After recommendations for device/channel reassignment and changing the resident SVC's, the measured time saved was 100 seconds per hour on each of the CPU's in the number of SVC loads and 88 seconds per hour per CPU by reduction in the number of seeks required for SVC loads. The total measured savings was 188 seconds per hour per CPU. Other recommendations were made and are being implemented.

SIDPERS, Ft. Riley.

Monitoring of SIDPERS Prototype Tests at Ft. Riley, Kansas, was the first trip into the field with the X-RAY Hardware Monitor. The X-RAY Hardware Monitor is approximately 5.5 feet tall, 2 feet square and weighs 460 pounds. Although it is mounted on casters and is classed as portable, unless there are elevators or ramps leading to the computer site, difficulty can arise in locating the monitor in the computer room. If this particular monitor is crated and shipped to the computer site by air freight, the memory unit of the monitor invariably becomes dislodged and must be reseated in its connector.

The SIDPERS Prototype Tests were conducted on an IBM S360/30 with 128K bytes of core and a single set of peripherals consisting of four IBM 2401 tape drives, four IBM 2314 disk units, one card reader/punch and a single printer. There were no vendor supplied monitor probe points for the IBM 360/30. Consequently, probe points had to be developed from the IBM logic diagrams before monitoring could be continued. This trip was extremely beneficial in revealing problem areas in the field shipment, installation and monitoring with a hardware monitor.

SAILS RUN TIME REDUCTION STUDY

GENERAL. A study designed to improve the performance capabilities of a Command standard system through the use of performance measurement and evaluation tools was conducted in June 1973. As the responsible agent for the development of multicommand standard software systems, any inefficiencies contained in USACSC developed software are multiplied many times over when the software is released for operation at numerous Army sites. Generally, programs are released to the field when they run in a specified hardware and software environment and satisfy specified functional requirements; performance goals are usually afforded a lower priority.

The Standard Army Intermediate Level Supply (SAILS) System, a multicommand, integrated automatic data processing system, was selected for a run time reduction study. This system was developed for implementation on IBM 360/30-40 systems under DOS. Although SAILS is a newly developed supply and related financial management system, selected modules from two other Army Supply Systems are included in the total package. As a result of this amalgamation, a fairly complex system has emerged, with 70 programs comprising the daily cycle. Some of the larger programs, with as many as 30,000 COBOL source statements, contain numerous overlays to accommodate the 98K core allocation constraint designed for the problem program. At the time of this study, the system was undergoing prototype testing and the basic cycle run times far exceeded expectations. In fact, it seemed apparent that approval for extension of SAILS to other sites would be contingent upon some streamlining of the system. In general, serious doubts were expressed as to whether the existing hardware at 41 Army bases could handle this additional workload.

Although the need for a run time reduction study surfaced, actually selling the idea of initiating a full scale study was met with some resistance. To understand this resistance, a brief review of some of the problems experienced is presented.

- Management Support. Difficulty was encountered in convincing management that allocating resources to an optimization effort was as important as supporting other priority commitments. Meeting present milestone schedules took precedence over efficiency of operation. The continual receipt of functional modifications also was frequently cited as reason for deferring the proposed study.

- Past Usage of Performance Measurement Tools. Until recently, the use of software performance monitors at USACSC was limited. For a long period of time, vendor releases of the software monitors either contained errors or when employed in a particular environment, presented some special problem. It was not always clear who was at fault; however, a fair assessment of the situation is that success was only obtained through a process of trial and error. The documentation, training and vendor support did not prepare the user for the "out of the ordinary" situation.

- Experience with Optimizing Techniques. Once the problems of getting the monitors operational were overcome, few personnel had the necessary training and experience to effectively analyze the trade-offs of implementing various optimizing alternatives. The development of ability to identify improvement potential and make experimental modifications took time.

However, once these problems were resolved, the study proceeded smoothly. The approach taken and results obtained follows.

Task Force Development. To accomplish this study in an expeditious manner, a team was formed whose members had the prerequisite skills and capabilities as described below.

- One individual proficient in the use of JCL and performing DOS system tasks (i.e., library functions, use and application of executive software); also knowledgeable in the COBOL language, programming techniques and hardware timing and performance characteristics.

- One individual knowledgeable in the functional aspects of the SAILS system as well as the ADP system functions (i.e., overall logic flow and processing concepts; file formats, organization and accessing methods; and interfaces with the remainder of the SAILS system or other external systems).

- One individual capable of operating the SAILS system on an IBM 360/30-40; also capable of handling the scheduling, management of tapes and disk files, and some statistics collection.

- Three individuals proficient in the use of performance monitors, their operation and implementation, as well as the interpretation of analysis data and subsequent implementation of various optimization techniques.

Baseline Cycle Selection. A baseline cycle was selected against which all subsequent testing was to be performed. Primary considerations in the selection of this cycle for monitoring, analysis, and trial modifications were:

- A cycle for which detail statistical and job accounting information was available.

- One which had processed a representative volume and mix of transactions.

- One which had processed the broadest possible range of transactions in terms of transaction type and logical processing situations. This qualification was to insure the highest degree of reliability that trial program modification which were to be made would not adversely affect functional logic.

A cycle processed during prototype testing, which met the above criteria, was selected. All master files, libraries and inputs necessary for rerunning in a controlled test environment were available. For accounting purposes, both console logs and Johnson Job Accounting reports were available.

Subject Program Selection. After giving appropriate consideration to the objectives of the effort, resources available and time limitations, it was obvious that initial efforts would have to be restricted. This restriction was in terms of both the number of programs to be studied and the degree of optimization attempted. For these reasons, only nine of the basic cycle programs were selected for study. These programs were selected on the basis of their average run time from statistics accumulated on all cycles processed during the prototype testing. The rationale used in selecting these programs was that they would provide the greatest potential for immediate reductions in cycle run time. Each program had averaged between one and six hours effective production time while the group as a whole was consuming approximately 60-70% of the total cycle time.

Object Computer for Experiment. Since most extension sites currently have S360/30's, it was recognized that the ideal environment for conducting the planned tests would be a S360/30 configuration with peripherals identical to those on which the system will run; however, dedicated time in the required quantity and time frame could not be obtained. Consequently, it was decided that a S360/40 with identical peripherals was the most suitable alternative. Since the nature of many actions to optimize a program are actually "tuning" it to a particular environment, results obtained on the S360/40 were later validated on a S360/30.

Tools Employed. Three proprietary software products were used to analyze and monitor the subject programs.

- o STAGE II - Tesdata Corporation. This package is a "COBOL" source code optimizer which accepts as input the source program statements and produces a series of reports which identifies possible inefficiencies and potential areas for improvement of both run time and core storage requirements.

- o DOS Problem Program Evaluator (DPPE) - Boole & Babbage, Inc. This product consists of two basic elements, the Extractor and Analyzer. The Extractor resides within the same partition as the object program during its execution. Statistical samples of activities within the object program are collected via a time based interrupt and recorded on a temporary file. The analyzer then uses this recorded information to produce reports and histograms showing percentage of CPU activity within the program's procedure at 32 byte intervals, quantifies both file oriented and non-file oriented wait time and identifies that to which it is attributable.

- o DOS Configuration Utilization Evaluator (DCUE) - Boole & Babbage, Inc. This software monitor also consists of an Extractor and an Analyzer. The techniques used are similar to DPPE, but in this case, resulting reports are relative to performance and loading of the hardware and system software as a whole. Detailed data is presented regarding such items as input and output device utilization, channels, disk arm movements and SVC library.

Experiment Constraints. In order to insure a reasonable degree of confidence that proven benefits would result and could be quantified with available time and resources, certain constraints were self-imposed. Only those program modifications which could be accomplished with three or four manhours of analysis and reprogramming effort were considered for immediate implementation and testing. Changes meeting this criteria were not made unless they could be accomplished within the confines of system resources presently used by the program. Specifically, no changes were considered which would require more than 98K core, more tape drives, additional disk space, or which would involve a major impact on the remainder of the SAILS system. These constraints assured that any significant identified savings could be incorporated and issued to the field with a minimum of further analysis. Caution was used so as not to make changes which would adversely affect the program's functional logic. Although potential for improvement was indicated in these areas, they were not addressed due to the risk of creating "bugs" in the programs and thereby, jeopardizing the objective.

Methodology. Preparatory to actual testing, arrangements were made with Tesdata Corporation to acquire STAGE II on a 30-day trial basis. STAGE II reports were then obtained for the nine selected programs. Initially, it was planned to proceed immediately with changes indicated by STAGE II and appraise results through monitoring as the first step in program testing. This course was subsequently abandoned; however, when two factors became apparent. First, several errors and limitations were discovered in the STAGE II reports which indicated the information was somewhat less than totally reliable.

Second, the volume of diagnostics created by STAGE II made selection of prime optimization areas difficult. As verified later by testing with DPPE, certain routines, although employing other than the most efficient techniques, simply were not executed frequently enough to consume significant amounts of time. Therefore, efforts were concentrated in accordance with DPPE and DCUE indications, using STAGE II reports as an aid in determining alternative methods and techniques to incorporate.

Initially, the entire benchmark cycle was run on the S360/40 in a controlled environment and all intermediate files necessary to test trial modifications for the nine selected programs were captured. Timing statistics were collected and execution times for each of the subject programs were recorded as the baseline time to which all comparisons would be made. All timed tests conducted throughout the effort were in a dedicated environment. Worthy of special note is the fact that these baseline times were obtained without a monitor in operation, while all times recorded for modified programs were obtained with a monitor in operation and are therefore inclusive of monitor overhead (estimated at no more than 5%). This means that reduction of run times resulting from subsequent testing of modified programs are, if anything, understated.

After completion of the benchmark cycle test, programs were modified on the basis of DPPE and DCUE reports, tested while remonitoring, and modified still further when this was feasible under the constraints mentioned earlier.

TEST PROFILE
RUN TIME, CORE USAGE, CPU ACTIVITY, WAIT TIME

		EXECUTION TIME		CORE STORAGE			WAIT	
PROGRAM	SUBJECT RUN	ELAPSED	ACCUM CHANGE	USED	ACCUM CHANGE	CPU ACTIVE	FOW	NON FOW
P08ALB	Baseline	27:10	NA	93,531	NA	80.00%	3.10%	10.28%
	Modified	25:05	-7.7%	89,903	-3.9%	85.12%	3.48%	11.40%
P69ALB	Baseline	21:33	NA	85,525	NA	84.46%	.89%	14.57%
	Modified	18:02	-16.3%	89,525	+4.7%	72.76%	5.48%	18.15%
P22ALB	Baseline	2:27:17	NA	85,651	NA	64.20%	28.14%	.58%
	Modified	2:06:43	-14.0%	91,859	+7.3%	67.78%	31.91%	.31%
P23ALB	Baseline	1:22:23	NA	89,951	NA	63.48%	35.58%	.90%
	Modified	1:16:40	-6.9%	97,723	+8.6%	65.59%	34.20%	.48%
P50ALB	Baseline	1:00:52	NA	85,525	NA	53.58%	37.33%	7.85%
	Modified	41:23	-32.0%	89,525	+4.7%	84.97%	1.90%	11.42%
P72ALB	Baseline	15:08	NA	82,095	NA	91.27%	1.71%	6.88%
	Modified	10:14	-32.4%	73,825	-10.1%	89.15%	3.68%	6.98%
P27ALB	Baseline	43:16	NA	12,098	NA	97.58%	1.06%	2.29%
	Modified	3:35	-91.7%	10,034	-17.1%	41.60%	38.94%	18.80%
P10ALF	Baseline	22:39	NA	69,751	NA	34.76%	62.95%	2.26%
	Modified	12:52	-43.2%	85,549	+22.7%	74.40%	20.61%	4.87%
P11ALF	Baseline	1:48:19	NA	96,761	NA	16.73%	10.51%	72.76%
	Modified	1:38:48	-8.8%	97,609	+.9%	- Not	Monitored -	

FIGURE 3 Summarization of Results

SUMMARIZATION OF RESULTS

	360/40 Benchmark Tests			360/30 Verification Tests		
PROGRAM	UNMODIFIED RUN TIME	MODIFIED RUN TIME	% REDUCTION	UNMODIFIED RUN TIME	MODIFIED RUN TIME	% REDUCTION
P08ALB	27:10	25:05	7.7%	47:36	45:50	3.7%
P69ALB	21:33	18:02	16:3%	35:31	26:50	24.5%
P22ALB	2:27:17	2:06:44	14.0%	3:46:30	3:16:47	13.1%
P23ALB	1:22:23	1:16:40	6.9%	2:01:00	1:52:46	6.8%
P50ALB	1:00:52	41:23	32.0%	1:28:38	1:07:19	24.1%
P72ALB	15:08	10:14	32.4%	22:06	15:07	31.6%
P27ALB	43:16	3:35	91.7%	1:30:05	3:27	96.2%
P10ALF	22:39	12:52	43.2%	36:46	21:01	42.8%
P11ALF	1:48:19	1:38:48	8.8%	2:06:28	2:05:13	0.6%
TOTAL	8:48:37	6:53:23	21.7%	13:14:40	10:14:20	22.7%

FIGURE 4 Test Profile - Run Time, Core Usage, CPU Activity, Wait Time

Verification that each modified program had performed correctly during the tests consisted of comparing input and output record counts between the benchmark test cycle and the respective program test. In addition to control count comparisons, one program was further verified by an automated tape comparison of output files between the baseline and test runs.

Analysis and Findings. A review of the selected programs revealed many cases of inefficient COBOL usage. It should be noted, however, that this situation is not unusual or unique to this particular system or organization. Very often, application programmers have little exposure to efficient COBOL implementation. Although numerous alternative optimizing techniques exist, for this study only a few of the many available ones were utilized. Only those techniques which were potential candidates for large scale benefits with a minimal expenditure of effort were employed. To assess the merits of various techniques and select the most appropriate ones, a knowledge of the efficiencies/inefficiencies inherent in the COBOL language, the compiler, the operating system and the hardware itself was applied. Major modification areas for this study included:

- Buffering and Blocking. In the SAILS system, core limitation necessitated single buffering several key files. Since this system runs in the background partition with only SPOOLing processed in the foreground, the use of double buffered files in place of single buffered files to reduce file oriented wait greatly improves total system performance. In those cases where core was available or could be made available, files were double buffered on a priority basis according to the amount of wait time indicated by DPPE. Reducing the block size of key files required an impact analysis of possible external interface problems or possible degradation to other programs in the SAILS system which also accessed the key files.

- Table Handling. Many tables processed by the SAILS system, were searched using iterative subscripted loops. To reduce program execution time, three alternative searching techniques were employed; serial search, partition scan, and binary search. Depending upon the number of entries and the table organization, the most appropriate one was selected.

- Subscript Usage. The misuse of subscripts was identified as a contributor to excessive run time. Techniques employed to either eliminate or reduce the number of subscripts in a particular program included serial coding, use of intermediate work areas, use of literal subscripts and combining multiple subscripts to a single level by redefining the matrix as one dimension and applying a mapping function.

- Data Patterns Analysis. Data patterns occurring within the problem program were identified and depending upon their frequency of execution were altered. The importance of testing with representative data cannot be overstated when tuning a system through the rearrangement of data patterns.

- Data Definition and Usage. Special attention was given to assigning the most efficient usage for data item representation in core storage for those cases where the applicable item had a significant impact on CPU activity. Also steps were taken to eliminate or at least reduce to a minimum mixed mode usage in both arithmetic expressions and data move statements.

- Console Messages. It was found that a considerable amount of time was being consumed printing console messages. Those messages that required no monitoring or response by the operator were diverted to the SYSLST in lieu of the console printer. To even further eliminate console printout, it was suggested that eventually the system be modified so that simple comparisons and checks of control counts become as integral part of the system logic and not the responsibility of an operator.

In general, the modificiations employed during this study were concentrated in the areas listed above. At Figure 3 is a summarization by program of both the baseline and modified program run time for the benchmark test on the IBM 360/40 and the verification test on the IBM 360/30. Notice that for the benchmark test the degree of success ranged from 6.9% to 91.7%. The net result was a 21.7% reduction in the processing time of the nine selected programs. Resources used to accomplish this effort were 114 man days and 135 hours of computer time. The chart at Figure 4 is a test profile showing run time, core usage, CPU activity and wait time for each program before and after modifications were made.

For three of the selected COBOL programs, some examples of the analysis, techniques employed and results achieved is presented.

P50ALB. This program originally ran 1:00:52. The initial DPPE analysis showed 37.33% of the total run time attributable to File Oriented Wait (FOW) of which 8:42 or 14.10% was wait time on the input Document History (DH) file and an additional 9:43 or 15.74% was wait time on the output DH file. The DH file contained variable length records ranging from 175 to 2100 characters with an original block size of 8400 characters. Due to core limitations, both the input and output files were single buffered. To reduce the high FOW, the DH file was reblocked from 8400 characters to 6300 characters; thereby, permitting both files to be double buffered. Active CPU time was recorded at 53.58% of the total run time. A review of the DPPE histogram showed two areas where a considerable amount of this time was being consumed. One of these areas was attributable to a routine which moved individual segments of the DH file to a work area via an iterative subscripted loop. A routine was substituted which accomplished the movement of these segments in mass. The second large time consuming area was identified as the processing time of two IBM modules used for handling variable length records. Since these modules, which accounted for 12% of the total processing time, were not accessible to the application programmer, no action was taken.

The first test after these modifications were made ran 46:12 which showed a 24% reduction from the unmodified version's run time. Total FOW time had decreased to 5:37 or 12.19% of the run time and the CPU time attributable to processing the DH segment move had decreased significantly. Still more areas for possible improvement were identified. Two single buffered files which previously had no FOW, now had wait time of 4:23 or 9.51% of the total run time. Consequently, both files were double buffered for the second test. After analyzing the impact of block size modification on 14 other programs using the DH file, it was decided to further decrease the block size to 4200 characters. By cutting the block size in half, all programs using the file could double buffer it without any alteration of core requirements. The second test ran 41:23, a net decrease of 23% from the unmodified versions run time. FOW had diminished to a level of 1.90% or less than a minute of the total run time. Active CPU time was reported as 84.97% of the total run time. Active CPU time was reported as 84.97% of the total run time. The IBM variable length record modules now accounted for 19% of the time. The remainder of CPU active time was spread across numerous small routines. Because of the amount of time required to research each item, additional changes, which could bear heavily on CPU active time, were not pursued during this study.

P22ALB. In the baseline cycle, this program ran 2:27:17. The initial DPPE analysis showed FOW at 28.14% of the total run time. The bulk of FOW was on disk resident master files, all of which are accessed by a Command written Data Management Routine (DMR). SELECT clauses, data descriptions and the actual reads and writes of these files are not part of the problem program and therefore, opportunities for optimization were limited. To reduce FOW, three single buffered tape files, which constituted 2.5% or 4 minutes of the wait time, were double buffered. Also some nonresponse type console messages were diverted from SYSLOG to SYSLST. Active CPU time was recorded at 64.20% or 1 hour, 21 minutes. A breakout of the CPU time revealed that 35.43% or 53 minutes was spent in an IBM Direct Access logic module. An analysis of the CPU active time spent in the problem program showed that the three most time-consuming routines accounted for only 1.94% of the total processing time. In other words, practically all processing which was affecting the run time was taking place outside the problem program. Therefore, only minor changes were made to improve computational efficiency. The first modified test ran 2:11:15 an 11% reduction from the baseline test. As already stated, the fact that the routines causing the most wait or most CPU activity fell outside the realm of the problem program, made short range modifications difficult. To help identify possible improvements in the area of disk organization, P22ALB was rerun using DCUE. Information obtained from the DCUE analysis was then used to reallocate disk extents in an attempt to reduce arm contention. The results of this second test, which ran in 2:06:43, showed only a 4 1/2 minute reduction. This reduction was somewhat less than expected. At this time, it became increasingly clear that the DMR currently in use was extremely costly in terms of total SAILS processing time. As a result, no additional tests were made and a major study of available file accessing routines was recommended.

P27ALB. During the baseline cycle run, the execution time of this program was 43:16; however, it had run as long as 6 hours during prototype testing. A review of the most frequently executed routines revealed that 95% or 40 minutes of the total processing time was spent in a single routine which was designed to clear an array of 324 counters via a subscripted loop, "PERFORMED VARYING." This routine was replaced by straight-line coding a series of conditional statements and moves to clear 9 counters. When P27ALB was tested with the new routine, the results were startling. It ran only 3 minutes, 35 seconds, a reduction of 92%. The routine which previously had caused the excessive run time (40 minutes) was reduced to a total of 35 seconds. This test provided a classic example of the resulting differences between simple straight forward coding and usage of the more complex COBOL options with all their inherent inefficiencies.

Conclusions:

This study was successful in reducing the processing time requirements for the SAILS system. It also provided a good example of effective utilization of performance monitors. At the completion of the study, management could actually see the practicality of using monitors. It also served as the catalyst to initiate action on some ADP related problems, specifically, the need for a tutorial document providing programer guidelines and conventions and the need for a reappraisal of the Command Data Management Routines. As verified by benchmark testing, the 21.7% average reduction in processing time for the nine programs studied actually eliminated three hours from the daily cycle processing time. This savings when multiplied by the number of systems affected (41 Army bases) represents a substantial savings to the Army.

An optimization effort such as the one just described is not a simple task. It requires extensive planning, encompassing a review of the entire system, both software and hardware. In order to operate in the most expeditious manner and perform the greatest service, an independent group of qualified personnel with training and experience in all phases of optimization is necessary. Provisions for a dedicated and controlled test environment assure the highest reliability and predictability of results. As evidenced by this study, a substantial investment and management commitment is necessary to successfully complete an optimization effort.

A COMPUTER DESIGN FOR MEASUREMENT--THE MONITOR REGISTER CONCEPT

Donald R. Deese

Federal Computer Performance Evaluation and Simulation Center

Introduction. This paper is divided into two parts - in the first part I will describe (on a fairly high-level basis) the Monitor Register Concept; in the second part, I will describe a specific implementation of the Monitor Register Concept within the TRIDENT Submarine Fire Control System.

I. DESCRIPTION OF MONITOR REGISTER CONCEPT.

A. Hardware Monitor Concept. First, a bit of background on the concept of hardware monitoring so that the desirability of the Monitor Register Concept will be apparent. (I will not try to describe the justification for computer performance monitoring or the reason for using a hardware monitor; these subjects are adequately addressed in numerous published work.)

From a simplistic view, the hardware monitoring concept is as illustrated in Slide 1. When using a hardware monitor, very small electronic devices (called sensors or probes) attach to test points or wire-wrap pins on the back planes of computer equipment. These electronic devices use a differential amplifier technique to detect voltage fluctuations at the locations to which they are attached. (The voltage fluctuations represent changes in the status of computer components - Busy/Not Busy, True/False, etc.) The signal state, as detected, is transmitted by the electronic device over a cable to the hardware monitor. The hardware monitor measures the signal in one of several ways: it counts the occurrance of a signal state (e.g., count instructions executed, count seek commands, etc.); it times the duration of a signal state (e.g. time CPU Busy, time Channel Busy, etc.); or, it captures the state of one or more signals (e.g., capture contents of Instruction Address Register, capture contents of Storage Data Register, etc.). Many variations on this basic concept are commercially available.

B. Operational Problems with Hardware Monitors. There are a number of operational problems (Slide 2) which have plagued hardware monitor users:

1. Locating Test Points. There are thousands of test points on a computer system. It is often difficult for users to determine which test point (or combination of test points) reflects precisely the signal desired for measurement.

2. Signals Not Available at Test Points. Some signals - especially those involved with more esoteric measurements - are not available at test points.

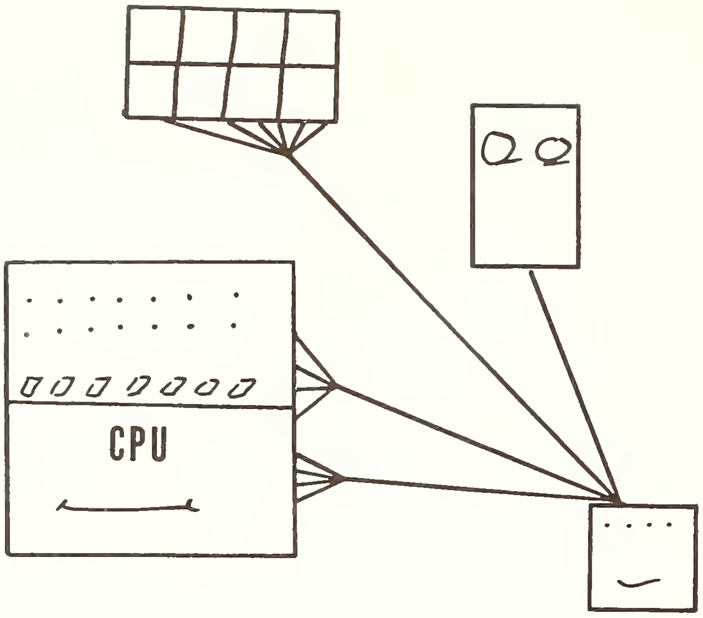
3. Signals Not Synchronized. As users of hardware monitors begin measuring "logical" aspects of their computer system (e.g. program activity, file activity, operating system activity, etc.), the simultaneous measurement of many signals is frequently required. The desired signals are not always synchronized; sophisticated measurement techniques are required to synchronize the signals - and frequently the measurement strategy must be abandoned because of synchronization failure.

4. Attaching probes. The process of attaching probes entails a risk of crashing the computer system and prudance requires a dedicated system; the process disrupts operation and alarms management.

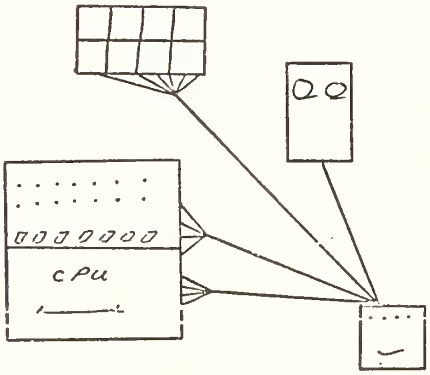
5. Changing probes. When measurement objectives change, often probes must be changed. The problems caused by changing probes are the same as initial probe attachment.

6. Large Number of probes required. An increasingly large number of probes is required with measurement strategies. This is especially true with the more sophisticated strategies which require the capture of data register or address register contents.

C. Monitor Register Concept. The Monitor Register Concept solves these problems!



o Slide 1 --- HARDWARE MONITORING CONCEPT



LOCATING TEST POINTS
SIGNALS NOT AVAILABLE AT TEST POINTS
SIGNALS NOT SYNCHRONIZED
ATTACHING PROBES
CHANGING PROBES
LARGE NUMBER OF PROBES OFTEN REQUIRED

o Slide 2 --- OPERATIONAL PROBLEMS

1. Data Flow. Before specifically discussing the Monitor Register Concept, let us consider the flow of data values within a computer system. Although actually quite complex, the data flow may be considerably simplified by considering only those registers containing the data or address values which reflect the internal operations of the computer. If the contents of these registers could be captured, and properly sequenced, the internal operation of the computer could be duplicated. (If these registers could be captured, all of the important computer functions could be measured, all the data could be captured, all addresses could be recorded, etc.)

Attaching hardware monitor sensors to each bit in these registers would require a large number of sensors (at least one sensor to each bit in every significant register) and the problem of signal synchronization would still exist.

2. Monitor Register. What is needed to solve these problems is some sort of funnel which would collect the contents of all of these registers-of-interest and funnel them to a common register which could be easily monitored. This is the essence of the Monitor Register Concept. As shown by Slide 3, with the Monitor Register Concept, the contents of a number of registers are collected and funneled to a single "interface" register. As each register value changes, the contents of the register are presented (one register at a time) to the interface register.

Of course, there really isn't a funnel inside computers. But there is something which is the electronic equivalent. I'll discuss that when I discuss a specific implementation of the Monitor Register Concept. To illustrate the concept, however, let us shrink the funnel in Slide 3 and place it inside a CPU (Slide 4) where it funnels register contents to a Monitor Register Interface. A hardware monitor can then be attached to this single common interface (Slide 5) and acquire the contents of all of the registers which are input to the "funnel."

The Monitor Register Concept eliminates the operational problems listed earlier. The test points are replaced with a specific Monitor Register Interface. All relevant signals are available at the Monitor Register Interface. The signals are synchronized when presented to the Monitor Register Interface. There is no need to change probes since all relevant signals are made available at the Monitor Register Interface, and, since all relevant signals are available at the Monitor Register Interface, a large number of probes is not required.

This is a simple concept. What about implementation of the concept into a specific computer design? The second part of this paper describes an implementation of the Monitor Register Concept under a project FEDSIM performed for the Naval Weapons Laboratory, Dahlgren, Virginia.

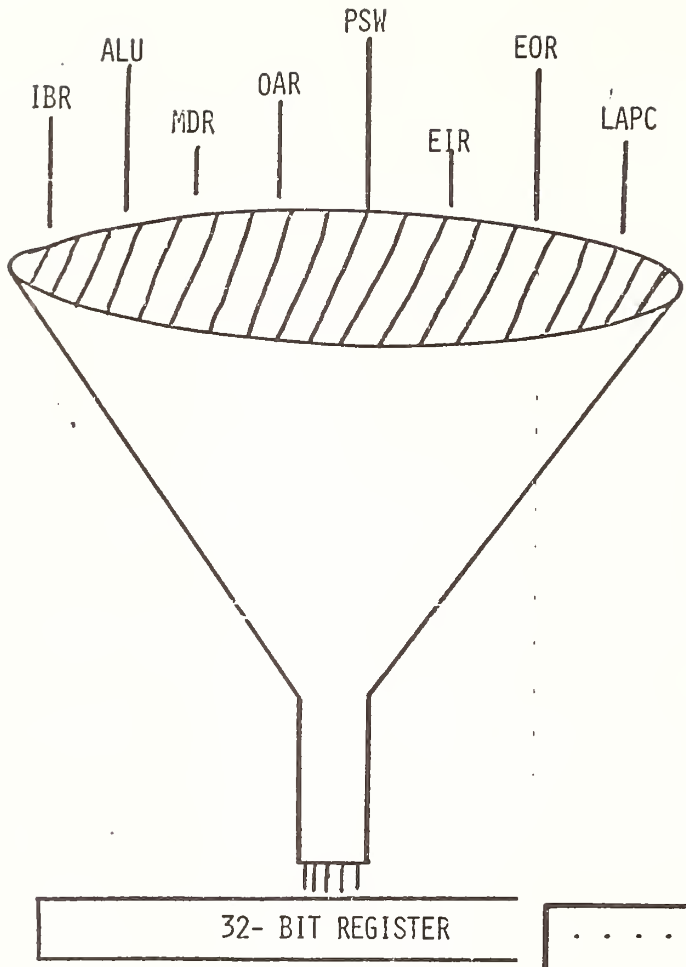
II. IMPLEMENTATION OF THE MONITOR REGISTER CONCEPT.

The Naval Weapons Laboratory (NWL) is located at Dahlgren, Virginia. One of NWL's Missions (Slide 6) is to develop and check-out the Fire Control System software for the TRIDENT submarine.

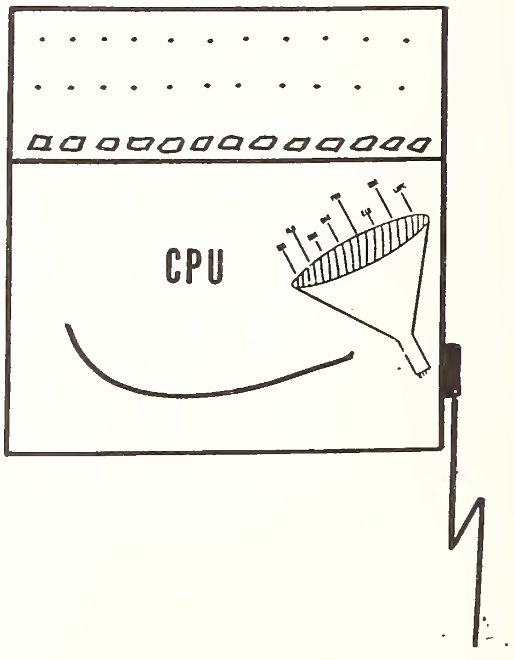
A. Fire Control System Software. The Fire Control System software must perform a number of complex tasks associated with the preparation for missile launch. Some of these tasks are computation of missile trajectory, missile alignment and erection computation, and guidance computer shadowing. These are just a few of the many functions performed by the Fire Control System software.

The Fire Control System software will operate in a real-time tactical system. This system has specific and critical time-dependent requirements; if the software does not respond to the external stimuli within a certain time, input data may be simply lost, or output may no longer be accepted. Further, real-time systems, by the very fact that time is a central factor, present unique problems. Real-time systems encounter unpredictable sequences of unrelated requests for system services. Since the timing interactions often are difficult to predict and may be impossible to consistently duplicate, program errors will frequently mysteriously and erratically appear.

Compounding the real-time problems facing NWL are the implications of the new systems for the TRIDENT submarine. When some problem occurs in the Fire Control System software, it may be difficult to determine whether the problem is caused by the application program or caused by the new computer system or the new operating system. The new computer system might have insufficiently documented features or pecularities, design problems, or actual hardware failure. The new operating system might have coding errors, logic errors, insufficiently documented features, and so forth. Even the assembler and compiler are new; these could generate



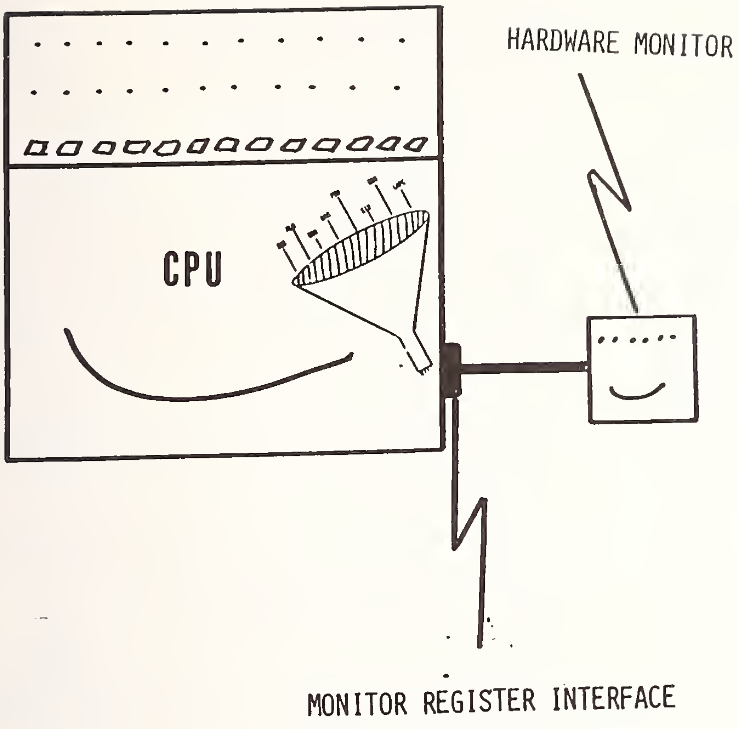
◦ Slide 3 --- MONITOR REGISTER CONCE

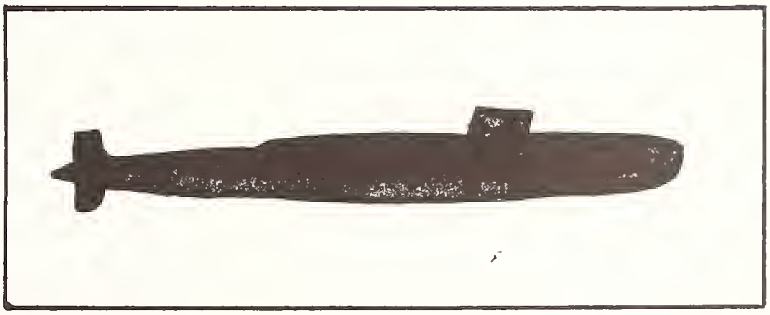


◦ Slide 4 --- MONITOR REGISTER INTERFACE

MONITOR REGISTER INTERFACE

◦ Slide 5





THE NAVAL WEAPONS LABORATORY

ONE MISSION

◦ Slide 6

DEVELOP AND CHECK-OUT FIRE CONTROL SOFTWARE erroneous code or data, they could have poorly documented features, etc.

In this difficult environment, NWL must develop the TRIDENT Fire Control System software. The software must undergo extensive test and debug. Since timing is often critical, the software must be efficiently tuned.

B. Performance Measurement of the TRIDENT Basic Processor. In this environment and with these tasks, NWL decided that performance measurement tools and techniques are needed. As a result, NWL asked FEDSIM to assist them with the design of appropriate performance measurement features for the TRIDENT Submarine Basic Processor.

As you might imagine, a nuclear submarine is not the typical place to find a hardware monitor. The environment poses some rather unique problems.

There are unique problems even when the Basic Processor is located in the Test Berth environment at the Naval Weapons Laboratory in Dahlgren. (For example, the computer will be ruggedized; it will be surrounded by the "bathtub enclosure" and there will be no possibility of connecting hardware monitor probes to the processor. If any signals are ever to be monitored, the signals must be made available to some external interface.) When we were considering the computer design approach to solving the problems of monitoring the TRIDENT Basic Processor, we quickly realized that the environment was ideal for the Monitor Register Concept.

C. Monitor Register Concept in the TRIDENT Submarine. As you know, computers do not really have a funnel into which data can be poured. However, the computer logic does have something very much like a funnel: the multiplexor. For the TRIDENT Basic Processor, we selected a multiplexor (Slide 7) having eight inputs and one output. Under microprogram control, at every minor cycle, the multiplexor selects one of the eight input registers and presents it to the output register.

The microprogram control is facilitated by five "performance monitoring" bits in the microprogram control word. At each minor cycle, three of these bits (value 000 to 111) are provided to the multiplexor. The multiplexor uses these three bits as a binary address to select the appropriate input register and provides the register bits, in parallel, to the Monitor Register.

The other two bits of the five in microprogram control word are used to identify certain unique characteristics of the specific data contained in the selected register. (For example, the Program Status Word (PSW) has a three-bit address of "001"; a two-bit characteristic code of "00" indicates a normal PSW, a code of "01" indicates an instruction-string-terminator PSW, a code of "10" indicates a successful-branch PSW, a code of "11" indicates an interrupt PSW.)

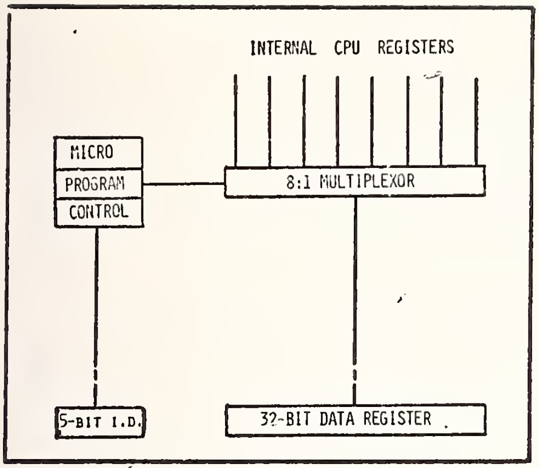
The programmer writing the microprogram does not have to continually worry about the "performance monitoring" bits in the microprogram control word; normally, these bits will be set to zero. The bits will be set to non-zero only at those few easily identified points in the microprogram routines where the appropriate registers are being altered.

The eight registers which are input to the multiplexor are identified in Slide 7. The Program Status Word provides the virtual instruction address, task identifier, condition designator, machine state, and interrupt status. The Memory Data Register provides operand values from either the memory or local processor registers. The Operand Address Register provides intermediate and final operand address values (both virtual and absolute). The Look-Ahead Program Counter provides the absolute instruction address. The Instruction Buffer Register provides the instruction word at the initial stage, intermediate stages and final stage of instruction execution. The Executive Input/Output Register provides the port and device address for all input and output. The Interrupt Register provides the interrupt level, interrupt conditions, etc. The CPU Timer Register provides the means of time stamping various events at the resolution of the Basic Processor minor cycle clock.

D. Uses of Data Available at Monitor Register. How can all of the data provided by these registers be used?

1. Program Debug (Slide 8). Practically all of the data provided at the Monitor Register could be used to debug programs. The Program Status Word could be used to debug programs. The Program Status Word could be used to trace program logic steps and module execution, and to identify non-executed program legs. The Operand Address Register could be used to identify references to specific program variables or data areas and the Memory Data Register could be used to analyze the values contained in the variables. Or, for example, the instruction address in the Program Status Word could be captured whenever selected variables were referenced. This could be used to determine which instructions were referencing certain variables.

2. Program Tuning (Slide 9). The data provided at the Monitor Register could be used to "tune" or optimize specific user programs. The Program Status Word could be used to obtain a distribution of



REGISTERS

PROGRAM STATUS WORD
MEMORY DATA REGISTER
OPERAND ADDRESS REGISTER
LOOK-AHEAD PROGRAM COUNTER
INSTRUCTION BUFFER REGISTER
EXECUTIVE INPUT/OUTPUT REGISTER
INTERRUPT REGISTER
CPU TIMER REGISTER

∘ Slide 7

∘ Slide 8

P R O G R A M D E B U G

- PROGRAM TRACE
- MODULE TRACE
- IDENTIFY NON-EXECUTED INSTRUCTION
- RECORD VALUES OF SELECTED VARIABLES
- RECORD ADDRESS OF INSTRUCTIONS REFERENCING SELECTED VARIABLES
- CREATE DETAILED PROGRAM PROFILE

∘ Slide 9

P R O G R A M T U N I N G

- CPU EXECUTION BY AREA OF CORE
- USE OF SUPERVISOR SERVICES
- PAGING ACTIVITY
- CODING TECHNIQUES
- INDEXING LEVELS

∘ Slide 10

I N S T R U C T I O N A N A L Y S I S

- OP-CODE ANALYSIS
- INSTRUCTION PATTERNS
- INSTRUCTION SEQUENCES
- INSTRUCTION STRING LENGTH ANALYSIS

COMPUTER COMPONENT MEASUREMENTS

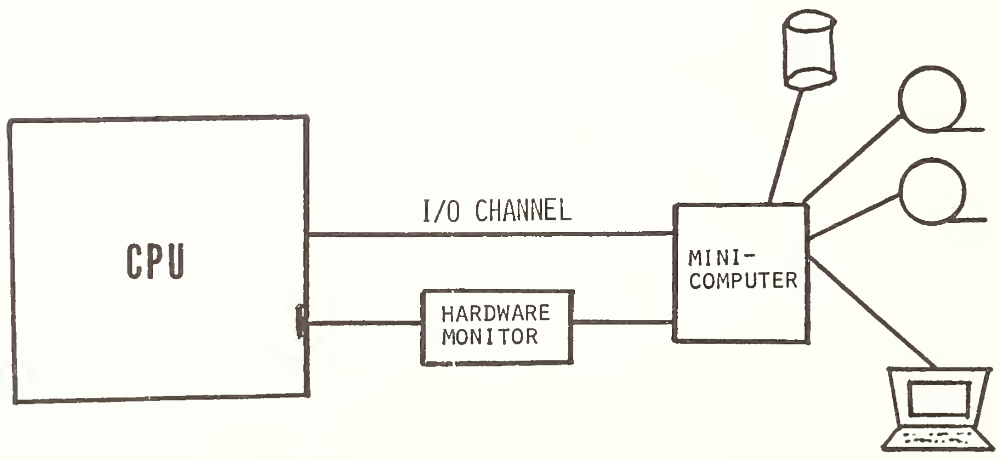
∘ Slide 11

- CPU BUSY/WAIT
- EXEC BUSY
- INPUT/OUTPUT TIMING, BY CHANNEL AND DEVICE
- DEVICE ERRORS
- INTERRUPT
- RACE CONDITIONS DURING PEAK I/O
- MEMORY CONFLICTS

∘ Slide 12

THE MAJOR POINT

EVERY SIGNIFICANT ADDRESS, INSTRUCTION OR DATA VALUE IS AVAILABLE AT THE MONITOR REGISTER INTERFACE. IF ALL THIS DATA WERE RECORDED, THE COMPUTER PROGRAMS' EXECUTION COULD BE SYNTHESIZED!



∘ Slide 13 --- TEST BERTH MODE — INTERACTIVE MONITORING instruction execution by area of core, so that highly used segments of code could be examined for optimization potential. The Instruction Buffer Register could be used to analyze programmer coding techniques, the use of supervisor services by module, and the number of levels of indirect indexing. These two registers could be used together to analyze paging activity.

3. Instruction Analysis (Slide 10). The data provided at the Monitor Register could be used to analyze the instructions executed by the computer. The Instruction Buffer Register contains the instruction as it was fetched from memory, and reflects any modification made to the instruction because of, for example, indirect addressing. This data could be used for op-code analysis, examining instruction patterns, analyzing instruction sequences, and so forth. The Program Status Word is uniquely identified whenever program control is transferred; this feature facilitates instruction string length analysis.

4. Computer Component Measurements (Slide 11). Finally, the data provided at the Monitor Register could be used to measure the utilization of the computer components. The Program Status Word could be used to measure the amount of time the CPU is in Busy or Wait States, and the amount of time that the CPU is executing in Executive Mode or in a User Program. The Executive Input/Output registers contain I/O channel and device addresses when input or output is initiated or terminated. Therefore it is easy to determine the channel and device busy times, overlap times, CPU Wait on Channel or Device, and so forth. The Interrupt Register could be used to determine interrupt levels, interrupt types, device errors, and so forth. A combination of registers associated with memory, along with the CPU time, could be used to determine memory conflicts.

These are just examples of the uses for data available at the Monitor Register. The major point (Slide 12) which these examples tried to illustrate, is that every significant address, instruction or data value is available at the Monitor Register. If all this data were recorded, the computer programs' execution could be synthesized!

For a normal monitoring session, it is unlikely that all of the possible data would be actually captured or analyzed. The massive amount of data would preclude this except for very unusual measurement objectives. Each monitoring session likely would be collecting data to solve a particular problem.

E. Test Berth Configuration. How is NWL going to use this design? Slide 13 is a simplification of the Software Development System components which will be in the Test Berth at Dalhgren.

A hardware monitor is attached to the Monitor Register and interfaces between the TRIDENT Basic Processor and a mini-computer. The hardware monitor provides a high-speed electronic filtering, decoding, comparison and buffering capability. The main reason for the hardware monitor is that the mini-computer would not, of course, keep up with the rapid flow of register data provided at the Monitor Register. The hardware monitor passes on to the mini-computer only the data required for a specific measurement objective. In this way, the mini-computer can effectively process the data it receives.

The mini-computer processes data from the hardware monitor in a number of ways, depending on the specific measurement objectives (program debug, program tuning, and so forth). The mini-computer has its own peripherals - tapes, disks, and a CRT; the mini-computer is connected directly to the TRIDENT Basic Processor by an I/O channel.

Programmers at NWL will use the mini-computer, via the CRT, for interactive monitoring. On command from a programmer, the mini-computer will load programs from its disks into the TRIDENT Basic Processor via the I/O channel. The mini-computer will set up control information for the Monitor Register logic and for the hardware monitor. Then, the mini-computer will initiate execution of the program just loaded. The programmer can have monitored data displayed on the CRT and/or have it written to one of the mini-computer's tapes for later analysis.

F. At-Sea Monitoring. When the TRIDENT submarine goes to sea, the Fire Control System software has been exhaustively debugged. However, we all know that software is never really completely debugged ("debug" means that all known errors have been corrected.); the best software usually eventually fails. Therefore, it is desirable to have some sort of monitoring capability at sea so that when the software does fail, the personnel at the Naval Weapons Laboratory have something to aid them in finding the error. This is especially important in that, since the software has been exhaustively debugged, error causes are difficult to determine.

There is a dual configuration in the TRIDENT submarine (to accomodate preventive maintenance, etc.). Slide 14 shows the implications of this dual configuration from a monitoring view - the two systems are cross-connected and monitor each other.

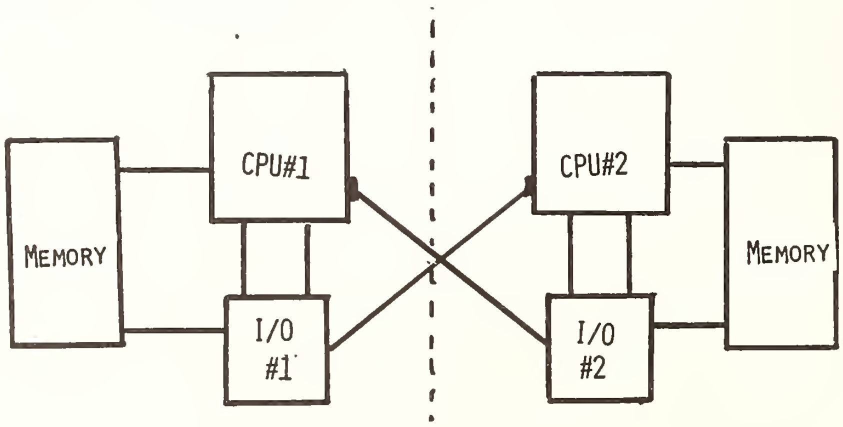
When the TRIDENT is configured for cross-connected monitoring, a port of the independent I/O control processor ("intelligent channel") would be connected to the Monitor Register of the other Basic Processor.

In the primary basic processor, control logic would be initialized such that only selected registers would be presented to the Monitor Register. In the secondary Basic Processor, a utility program would be loaded, request allocation of memory, and issue a Read I/O to its I/O port connected to the Monitor Register of the primary Basic Processor. Conceptually, the Read I/O would have two Data Control Words (DCWs) chained together. The first DCW would be a Read I/O operation with a data length the size of the memory allocated to the utility program. The second DCW would be a transfer to the first DCW. In operation, data would be read from the Monitor Register of the primary Basic Processor and placed in the memory area. When the memory area was full, the second DCW would transfer I/O control to the first DCW to begin the Read I/O again. The memory area would serve as a circular queue containing history data from the primary Basic Processor Monitor Register. Whenever some problem occurred with the primary Basic Processor, the memory in the back-up Basic Processor would be dumped to some external device for transmission to the Naval Weapons Laboratory. (As an illustration of this method, the utility program might be allocated 50,000 words of memory; control logic would be initialized such that only successful branch operation addresses would be presented to the primary Basic Processor Monitor Register; the history data available would be a record of the last 50,000 program logic legs executed.)

If information about the performance of future systems is to be available to the "outside world," specific design is required to make the information available.

The Monitor Register concept is one method whereby a variety of information about the internal computer performance can be made available at a common location and in an easily defined interface mechanism. I believe that the Monitor Register Concept has a place in future computer systems.

◦ Slide 14 --- TACTICAL MODE — CROSS-CONNECTED MONITORING



THE USE OF SIMULATION IN THE SOLUTION OF HARDWARE ALLOCATION PROBLEMS

Major W. Andrew Hesser, USMC

Headquarters, U.S. Marine Corps

INTRODUCTION

During the past year, Marine Corps personnel have been concerned with the redistribution of Marine Corps owned computer hardware at five major installations. This paper discusses the results of simulations of these installations.

All simulation models involved were created directly from hardware monitor data gathered with the SUM. The simulation language is a proprietary product, SAM, leased by the Marine Corps; the use of SAM in constructing simulation models was presented at the Spring, 1973 CPEUG Conference.

THE PROBLEM

The problems facing the Marine Corps were:

(1) A major installation presently has two System/360 Model 40's. QUESTION: Can the workloads of both be combined onto a System/360 Model 65?, a Model 50? Will there be sufficient capacity on each to process proposed wargames?

(2) A System/360 Model 65I was procured for the use of the operating forces of the Marine Corps in the Far East. The present computer, also a Model 65I, will be used only for garrison forces and all jobs currently being processed for the Fleet Marine Forces (about 60% of the present workload) will be shifted to the new computer. QUESTION: What size computer will be needed to process the remaining workload?

(3) The Marine Corps central supply center is being closed and relocated to another installation. The center has two System/360 Model 50I's, and the new location presently has a Model 50I. QUESTION: Can a Model 65J process the workloads of all three Model 50's combined? What size Model 65 can process the workload of the two Model 50's at the present supply center?

These questions were answered using simulation models.

QUANTICO

There are two installations at Quantico, Virginia, each of which has a System/360 Model 40. In October of 1972, a hardware monitoring team measured the Model 40 HGF with 448K bytes of core memory at the Computer Sciences School. Peripherals included eight CD-12 (2314-type) disk drives and six 2420 tape drives. Table 1 contains the results of the system measurement and the simulation validation.

TABLE 1
CSS QUANTICO VALIDATION

ACTIVITY	MEASURED	SIMULATED
CPU ACTIVE	76.1%	78.2%
SUPERVISOR	48.7%	51.0%
PROGRAM	27.4%	27.2%
CHANNEL 1	38.3%	37.7%
CHANNEL 2	6.0%	7.0%
MPX	.4%	.3%
TIME	160 MIN	160.1 MIN

In early 1973, another team measured the Automated Services Center at Quantico. This Model 40 HG with 384K bytes of core memory has eight 2314-type disks and eight tape drives. Table 2 contains the results of the system measurement and the validated simulation results.

TABLE 2
ASC QUANTICO VALIDATION

ACTIVITY	MEASURED	SIMULATED
CPU ACTIVE	70.5%	69.1%
SUPERVISOR	54.3%	52.5%
PROGRAM	16.2%	16.6%
CHANNEL 1 (DISK)	46.9%	45.6%
CHANNEL 2 (TAPE)	6.8%	6.8%
MPX	.6%	.8%
TIME	150 MIN	149 MIN

To determine what size system would be required to process the combined workload of the two installations, simulations were done on a Model 50I, then a Model 65I. The simulated configuration contained two banks of eight disks and one of eight tapes. The simulation results are contained in Table 3. The $T_j \neq 0$ notation means synthetic jobs arrived throughout the simulation. $T_j=0$ implies all jobs were in the job queue at time zero.

TABLE 3
COMBINED WORKLOADS OF THE CSS AND ASC

ACTIVITY	MODEL 50I $T_j \neq 0$	MODEL 65I $T_j \neq 0$	MODEL 65I $T_j=0$
CPU ACTIVE	45.7%	17.6%	34.0%
SUPERVISOR	30.6%	12.2%	22.3%
PROGRAM	15.1%	5.4%	10.7%
CHANNEL 1 (D)	33.0%	33.3%	67.7%
CHANNEL 2 (D)	30.4%	30.7%	60.2%
CHANNEL 3 (T)	13.7%	13.9%	27.3%
MPX	1.4%	.5%	1.1%
TIME	148 MIN	147 MIN	76 MIN

It was concluded from these results that a Model 50I CPU would be adequate for processing the combined Model 40HG and Model 40 HGF workloads. There would be unused CPU capacity available for processing small war games. A Model 65I would have considerable unused capacity available for processing large scale war games. However, additional processor storage would be required for either 512K byte machine if war games were to be added.

OKINAWA

During November of 1972, the existing system at the Automated Services Center on Okinawa was measured. The system consisted of a System/360 Model 651, (512K bytes of core storage), with two banks of eight disk drives and two banks of eight tapes on four separate channels. The measurement and validation results are contained in Table 4.

TABLE 4
OKINAWA VALIDATION

ACTIVITY	MEASURED	SIMULATED
CPU ACTIVE	24.4%	23.8%
SUPERVISOR	14.2%	14.2%
PROGRAM	10.2%	9.6%
CHANNEL 1 (D)	34.4%	41.3%
CHANNEL 2 (D)	26.5%	21.5%
CHANNEL 3 (T)	32.5%	28.2%
CHANNEL 4 (T)	25.4%	29.0%
MPX	1.7%	1.3%
TIME	180 MIN	182 MIN

There were four approaches taken toward the problem of matching a hypothetical computer to the workload which would be reduced by an estimated factor of 60%.

(1) Simulate using the total SUM measurement figures (entire existing workload).

(2) Simulate with the SUM results reduced by some constant percentage proportionate to the amount of workload which was to be diverted to the new computer.

(3) Simulate with the present SUM results reduced in a more sophisticated manner to reflect probable changes in processing characteristics.

(4) 'Conventional' simulation using SAM Language to build detailed models of all remaining processes.

For all simulations the currently installed peripherals were attached to the simulated CPU. The results of simulating the total workload on a Model 501 are shown in Table 5.

TABLE 5
SIMULATION OF 100% WORKLOAD ON A MODEL 501

ACTIVITY	$T_j \neq 0$	$T_j = 0$
CPU ACTIVE	64.1%	66.8%
SUPERVISOR	38.7%	39.3%
PROGRAM	25.4%	27.5%
CHANNEL 1 (DISK)	30.2%	32.8%
CHANNEL 2 (DISK)	39.9%	43.7%
CHANNEL 3 (TAPE)	24.9%	27.1%
MPX	2.9%	3.2%
TIME	206 MIN	189 MIN

The results show that the Model 50I would be capable of handling the total present workload within almost the same amount of time the Model 65I requires now. Considering that a highly active period was selected for input, the Model 50I certainly would be able to process the reduced workload.

The next two approaches were used to simulate the workload on a Model 40HGF (448K bytes) with two channels. Channel one had all sixteen 2314-type disks, and channel two had the sixteen 2420 tape drives.

The Data Processing Officer at the installation verbally indicated that the removal of jobs would leave about 40% of the total workload to be processed at the Automated Services Center (ASC). Using this information, models of the remaining ASC workload were created by reducing each hardware monitor input by 60%. Thus, each synthetic job made CPU and channel demands on the system equal to 40% of the originally measured data. It was realized that this did not represent the real world situation because some activity (like disk) may be reduced more than 60%. Nevertheless this method would provide an estimate of the effect of removing the Fleet Marine Force processing.

The ASC provided SMF statistics for all jobs processed during September and October 1972. This data, coupled with a listing of the Job Names of all jobs that would remain at the ASC, was used to reduce the SUM data that created the synthetic jobs. In this reduction instead of a 60% reduction in all activity, each of CPU, Disk, Tape and MPX activity was reduced in accordance with the actual amount of activity that would be shifted as determined by SMF.

The formula for the reduction percentage is shown in Table 6.

TABLE 6
SMF REDUCTION TECHNIQUE

$$\%\ \text{CPU ACTIVITY REMAINING} = \frac{\text{ASC JOB CPU TIME}}{\text{ASC CPU TIME} + \text{FMF CPU TIME}}$$

$$\%\ \text{CHANNEL ACTIVITY REMAINING} = \frac{\text{ASC ACCESSES}}{\text{ASC ACCESSES} + \text{FMF ACCESSES}}$$

The final reduction figures were:

(1) CPU activity reduced	67.7%
(2) Disk activity reduced	68.2%
(3) Tape activity reduced	65.2%
(4) MPX activity reduced	72.6%

Considering these figures with the 60% reduction above, it appears that the estimation of the Data Processing Officer was high for the processing that would remain at the ASC. The results of these two techniques are summarized in Table 7.

TABLE 7
SIMULATION OF REDUCED WORKLOAD ON MODEL 40HGF

ACTIVITY	60% REDUCTION	SMF REDUCTION
CPU ACTIVE	90.6%	79.9%
SUPERVISOR	58.8%	53.2%
PROGRAM	31.8%	26.7%
CHANNEL 1 (DISK)	23.4%	18.8%
CHANNEL 2 (TAPE)	21.4%	19.0%
MPX	1.3%	.9%
TIME	193 MIN	187 MIN

It is concluded that the Model 40 could process the reduced workload in about the same time that is required to process the total workload on the Model 50I now. The system would be CPU bound and would have little excess capacity for growth or contingencies.

The final step in the simulation analysis of the ASC workload was to be the construction of a model of each program that would be processed at the ASC. The bases for the models were the AUTOFLOW charts of the individual programs. Personnel at the ASC also provided probabilities of transaction flows for major decision points and file sizes and characteristics for the files used by the programs. Models were then constructed in the SAM language using detailed coding techniques.

The final simulations of each of the daily models were not completed due to software problems, excessively large (700K bytes) core requirements, and limited computer time. The models of the individual systems however can be used in future simulation applications.

The overall conclusions of the Okinawa study were that a Model 40 HGF would be adequate for processing the reduced workload but there would be little excess capacity for growth or contingencies. A Model 50I could process the reduced workload without difficulty.

PHILADELPHIA AND ALBANY

In June of this year, a monitoring team visited the Marine Corps Supply Activity at Philadelphia. The installation had two System/360 Model 50I CPU's which share most of the eighty-eight disk drives and twenty-two tape drives. The measurement and simulation validation figures are summarized in Table 8 below.

TABLE 8
PHILADELPHIA VALIDATION

ACTIVITY	MEASURED CPU "A"	SIMULATED CPU "A"	MEASURED CPU "B"	SIMULATED CPU "B"
CPU ACTIVE	63.4%	59.6%	90.1%	87.9%
PROGRAM	22.5%	22.4%	21.3%	20.3%
SUPERVISOR	40.9%	37.2%	68.8%	67.6%
CHANNEL 1	29.5%	16.6%	23.9%	29.5%
CHANNEL 2	51.1%	51.5%	13.2%	13.1%
CHANNEL 3	8.7%	20.2%	29.7%	26.8%
MPX	.9%	1.2%	.9%	.8%
TIME (SECONDS)	14400	14550	14400	15222

Simulations were then done on two Model 65's, an IH (778K) and a J (1024K). Each had two selector subchannels and 3 selector channels. The twenty-two tapes were accessible via either selector subchannel and all disk control units had a two channel switch. The results of the simulation using the combined workloads are summarized in Table 9 below.

TABLE 9
PHILADELPHIA SIMULATION

ACTIVITY	MODEL 65J $T_j \neq 0$	MODEL 65IH $T_j \neq 0$	MODEL 65IH $T_j = 0$
CPU ACTIVE	47.1%	47.1%	60.5%
PROGRAM	15.6%	15.6%	20.0%
SUPERVISOR	31.5%	31.5%	40.5%
CHANNEL 1	45.2%	54.6%	61.7%
CHANNEL 2	22.1%	12.7%	24.8%
CHANNEL 3	30.2%	32.0%	40.0%
CHANNEL 4	27.8%	26.4%	33.7%

TABLE 9 (CONTINUED)

ACTIVITY	MODEL 65J $T_j \neq 0$	MODEL 65IH $T_j \neq 0$	MODEL 65IH $T_j = 0$
CHANNEL 5	31.6%	31.2%	41.3%
MPX	2.4%	2.4%	3.0%
TIME (SECONDS)	14352	14356	11164

These results indicate:

(1) That with jobs arriving throughout the interval, the larger 1024K byte Model 65J has no advantage over the 768K byte Model 65IH. Both machines could process four hours of input data without difficulty in the same time. The activity levels of system components do not indicate any serious processing bottlenecks. It is noted that Channel 1 is the primary data path to the 22 tape drives and is used by the model whenever it is free. The channel imbalance between channels 1 and 2 therefore is of little significance.

(2) That the Model 65IH would have some excess capacity available as indicated by the results of having four hours of synthetic jobs all queued at the start of the simulation. The $T_j=0$ simulation does not consider job and job step interdependencies and file contention, however, and therefore represents the shortest possible processing time. Nevertheless, even with all partitions filled and jobs making maximum use of the system resources, there are no significant system bottlenecks. This simulation indicates that the Model 65IH processing the combined workload would not be a saturated system.

In February of 1973, a team of analysts measured the computer system at the Marine Corps Supply Center, Albany, Georgia. The Model 50I at that installation has forty disk drives supported by a two channel switch between channels one and two. Channels two and three have a total of fifteen tape drives. Table 10 below summarizes the results of the measurement and the simulation validation.

TABLE 10
ALBANY VALIDATION

ACTIVITY	MEASURED	SIMULATED
CPU ACTIVE	72.0%	69.6%
SUPERVISOR	53.2%	50.0%
PROGRAM	18.7%	19.5%
CHANNEL 1	53.5%	48.1%
CHANNEL 2	34.7%	48.4%
CHANNEL 3	6.1%	9.6%
MPX	1.1%	1.6%
TIME (SECONDS)	14400	14093

The workloads of Philadelphia and Albany were then combined into one workload to be simulated on a Model 65J with jobs arriving throughout and with jobs queued. The results of the two runs are summarized in Table 11.

TABLE 11
COMBINED ALBANY AND PHILADELPHIA

ACTIVITY	$T_j \neq 0$	$T_j = 0$
CPU ACTIVE	70.7%	74.7%
SUPERVISOR	48.5%	51.3%
PROGRAM	22.2%	23.4%

TABLE 11 (CONTINUED

ACTIVITY	$T_j \neq 0$	$T_j = 0$
CHANNEL 1 (T)	61.2%	60.6%
CHANNEL 2 (T)	34.0%	40.1%
CHANNEL 3 (D)	58.1%	57.0%
CHANNEL 4 (D)	55.5%	62.7%
CHANNEL 5 (D)	60.2%	63.9%
MULTIPLEXOR	3.8%	4.1%
TIME (SECONDS)	14406	13630

These results indicate that a System/360 Model 65J would be saturated if it were to try to process the total combined workload. The disk channel activity is higher than could be reasonably expected and is higher than any total channel activity ever measured in the Marine Corps. For example, the activity on the HQMC disk channels seldom exceeds 40 percent. The simulation with all jobs queued shows only a 6% improvement over the $T_j \neq 0$ simulation with regards to throughput time. It is emphasized that this is the best that could ever be expected and clearly indicates a saturated system.

RECOMMENDATIONS

There were four events recommended for 1973 and 1976. For 1973, the first event was the actual installation of the Model 65I at the Third Marine Amphibious Force on Okinawa. Next was the shifting of all processing presently at Camp Butler, Okinawa to the III MAF system. The third event was to ship the Model 65I CPU/CHANNELS from Camp Butler, Okinawa to the Marine Corps Supply Activity, Philadelphia and install the Model 65I CPU/CHANNELS with no other changes. One of the Model 50I's would then be moved from Philadelphia to Okinawa and installed.

In 1976, the target date for most of the changes, the four events would be as follows: First, shift some processing from Philadelphia to Albany, Georgia; then ship the System/360 Model 65I with ½ of the peripherals to Albany and install it with 256K additional core. The Model 50I at Philadelphia would continue to process. The third event for the year would be the shifting of the remainder of the processing from Philadelphia to Albany, Finally, the remaining Model 50I from Philadelphia could be installed at Quantico, Virginia. This would then release the two Model 40's there for re-utilization elsewhere.

SUMMARY

The Marine Corps has been able to use simulation as an effective tool for determining equipment allocation. The capability of building models from hardware monitor data has allowed models to be built in a short period when a considerably longer time would have been required otherwise.

HUMAN FACTORS IN COMPUTER PERFORMANCE ANALYSES

A. C. (Toni) Shetler

The Rand Corporation

ABSTRACT

The results of computer performance experiments executed in a normal work environment can be invalidated by unexpected activities of the people involved (operators, users, and system personnel). Human behavior during an investigation can confound the most careful analyst's results unless this behavior is considered during experimental planning, text execution, and data analysis. Human actions need not thwart efforts and can, in some instances, be used profitably.

In general, an analyst can design an experiment that precludes dysfunctional human behavior, can collect data to indicate when it occurs, or can plan the experiment so the behavior is useful.

INTRODUCTION

This paper considers human behavior during experimental planning, test execution, and data analysis for computer performance evaluation. Human problems in the testing environment during a computer performance evaluation effort can be likened to a field test for a product verified in a laboratory. That is, a hypothesis that explains user behavior or system relationships is usually first examined in a controlled environment. Once a hypothesis has been explained in the controlled environment, it is usually necessary to expand the explanation to the normal environment; this involves exposing the hypothesis to the variability of human reactions. A computer performance experiment using the normal system environment requires that an analyst be aware of the human problems associated with testing hypotheses.

Five suggestions to consider when conducting such experiments include:

- Ensuring participant awareness.
- Defining the measures.
- Testing for objectivity.
- Validating the environment.
- Using multiple criteria.

The format of this paper is to identify each consideration and present a brief illustration.

ENSURING PARTICIPANT AWARENESS

When an experiment relies on overt actions of people other than the analyst, it is critical that their orientation include an awareness of the experiment's relationship to the hypotheses; lack of awareness can lead to inconclusive results. The negative effect was clearly illustrated in an experiment where normal operations were required to validate system specific hypotheses.

The investigation involved increasing the peripheral capacity of an IBM 360/65 MVT system. The workload processed by the computer system indicated that projected workloads would saturate the system. A channel was available to attach additional peripheral devices to validate a hypothesis that suggested a solution to this problem.

The hypothesis was stated as follows: Adding peripheral devices to the available channel and directing spooled data to those devices will result in an increased capacity; proving and implementing this solution should result in an increased machine capacity. The experiment included 1) testing the hypothesis using a specialized job stream that led to stating the hypothesis; 2) if these controlled tests were positive, following them with a modified environment for normal operations; and 3) then verifying a capacity increase by examining accounting data. This test could only be executed once because a special device hookup was required which was borrowed from another computer system and had to be returned at the conclusion of the experiment - the time frame was a 15-hour period.

The first part of the experiment, to verify the hypothesis in a controlled environment, was dramatic in its conclusiveness. Therefore, we prepared for the second part, to validate the new peripheral configuration the following morning. The operators had modified start-up procedures and were told an experiment required them to use these procedures until 10:00 o'clock. The users were not informed of the experiment because their participation included submitting jobs -- a task they performed normally. The results were examined after the 10:00 o'clock completion and were very positive. In fact, substantially more work, in terms of jobs processed and computer resources used (CPU time, I/O executions, etc) had been accomplished with the modified peripheral configuration when compared to the equivalent time period of the previous week (morning start-up to 10:00 AM). Only later did we discover that the operators, in an attempt to "help" the experiment, got started much faster than they had the previous week and thus made the system available for 25% more elapsed time than is usual during the time prior to 10:00 AM. The test results were invalidated; nothing could be said about the effect of a modified peripheral configuration for the normal environment, and the addition of peripheral devices could only be based on the non-system specific testing in the controlled environment using the test job stream.

This experience most emphatically demonstrated the problem of active participants not being aware of their role in the experiment -- help from the operators invalidated the experiment. This problem is termed the "Hawthorne effect", and it must be considered whenever it may invalidate results.

DEFINING THE MEASURES

When using people to measure the results of an experiment, the measures should be selected to require the least overt action on their part. A particularly vivid illustration of this occurred in an experiment to test the resolution of priorities among on-line users.

The on-line user jobs locked each other out for long periods (5-10 minutes) and these users were extremely vocal in their concern that something be done to eliminate the lockout. The problem resolution constraints were:

- o No user was to be singled out as the culprit and prevented from executing;
- o No user program should require modification to implement a solution;
- o No job could lockout other jobs for a prolonged period (greater than 5 to 10 seconds);
- o The solution must be simple to implement and inexpensive to run -- the available time-slicing facilities were not acceptable.

The measure defined was quite simple: the intensity of the on-line users' complaints about lockouts -- these users would complain until their problem was solved. The procedure for testing the hypothesized solution was also very simple -- a system modification to the interrupt logic that rotated jobs of the same priority preventing any one job from gaining control for a prolonged time. The system modification was first tested in a controlled environment with special tests to verify its execution characteristics, and was then used for several days in the normal work environment.

Aside from verifying that on-line users were actively using the system (through accounting and observation), the validity of the hypothesis was confirmed by noting the abrupt end to the complaints about lockout. This experiment emphasized the need for an appropriate measure that was easy to collect from the users.

TESTING FOR OBJECTIVITY

Users are not always good subjective evaluators of system changes, and the analyst should not trust them. An experiment to evaluate the effects of reducing the memory dedicated to an on-line system required soliciting user evaluation to identify the effect of reducing the available memory. The users were told that reduced memory configuration would be available Monday, Wednesday, and Friday, and standard memory would be available Tuesday and Thursday. Following the experiments, the users claimed that the Monday/Wednesday/ Friday service was unacceptable and the Tuesday/Thursday service was very good. Only subsequently were they informed that Monday/Wednesday/Friday service was based on the reduced memory configuration. Self-interest, combined with changed perceptions of reality, make users poor subjective evaluators of system modifications.

VALIDATING THE ENVIRONMENT

Test periods can coincide with a particular event and can invalidate the experiment. Spurious workload changes can also invalidate experimental results. If a testing period coincides with a particular event, such as annual accounting and reports, the results can be deceiving when applied to the standard working environment. Normal fluctuations in workload over short periods (one or two days) can result in the same effect.

One short test (one-half day) was seriously affected by a user who had just completed debugging several simulations and submitted them for execution. The execution of these simulations resulted in extraordinary CPU utilization (approximately double), definitely out of proportion to the workload characteristics when data over a longer period were examined. Fortunately, this workload shift was discovered, though many analysts forget that a computer system seldom reaches a "steady state". The workload characteristics of a system during a short test period must be compared with characteristics over a longer period. A formal comparison period with appropriate controls should be used to reduce the probability of invalid results caused by autonomous workload changes.

Accounting data should be verified to validate the normalcy of the environment. The measures that can be included are:

CPU utilization

Mass store channel utilization

Tape channel utilization

Memory requested per activity

Memory-seconds used per second

Average number of activities per job

Activities initiated per hour

Tape and disk files allocated per activity

Average number of aborts per 100 activities

Cards read and punched per activity

Lines printed per activity

If the objectives, hypotheses, and procedures for an experiment are clearly defined, many of the metrics listed above may be irrelevant and the analyst can ignore them.

USING MULTIPLE CRITERIA

Validating hypotheses about major changes to the computer system presents a serious problem. The magnitude and direction of the proposed modification usually dictate the level of investigation to the impact on the users. This level is usually quite high, particularly if the objective of an investigation is to reduce the availability of resources. People appear to accept results more readily when a number of different indicators all point toward the same conclusion.

An investigation to remove a memory module, at a savings of $7,500.00 per month was definitely desirable from the computer center's viewpoint if the following could be validated:

- o Batch turnaround would not increase beyond one hour for the majority of the jobs.
- o On-line system response time would not increase significantly.
- o The quality of response to on-line systems would permit them to still be viable.

The experiment involved collecting measures from the system, systems personnel, and a hardware stimulator. Measurements from a three week period were analyzed. The test plan called for the first and third week to be used as a control period, while the second week provided a test environment that simulated the less expensive hardware configuration. The results of the experiment were:

- o Batch turnaround had increased from 0.5 to 0.7 hours, with 60-80%, rather than 90% of the work being accomplished within an hour (determined from statistics on batch turnaround).
- o On-line system response time had no statistically significant change; a variety of on-line system commands were tested using a special purpose hardware stimulator.
- o BIOMOD, a full vector graphics system, was used by systems personnel to evaluate the quality of the on-line response -- the quality was found to be degraded, but not disabled. The users of these services could still function.
- o CPU utilization had gone up 5-10% above the 60-65% level it had been operating at previously (data from system accounting).

By combining the different criteria it was established, with reasonable certainty, that the memory module could be removed without imposing on the productivity of the users of that system. Virtually no one complained that the conclusion was incorrect although several users (including one experimentor) vehemently argued against the core reduction until the multiplicity of unanomous indicators was known.

CONCLUSIONS

Introducing human variability into the arena of computer performance evaluation experimentation requires that the analyst take more than the usual precautions, as illustrated above. In addition, the general procedures to validate hypotheses about a particular system requiring testing in the uncontrolled environment should include verifying the hypotheses in a controlled environment and then, exposing the hypotheses to the uncontrolled environment. The normalcy of the environment should be validated through checks on the accounting data and through use of a control period, as appropriate. The inclusions of these should result in more accuracy in testing hypotheses involving people interactions.

REFERENCES

Bell, T.E., B.W. Boehm, and R.A. Watson, "Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort," The Rand Corp., R-549-1-PR, November, 1972.

Bell, T.E., "Computer Measurement and Evaluation -- Artistry, or Science?", The Rand Corp., P-4888, August, 1972.

Bell, T.E., "Computer Performance Analysis: Measurement Objectives and Tools," The Rand Corp., R-584-NASA/PR, February, 1971.

Kolence, K.W., "A Software View of Measurement Tools," DATAMATION, Vol. 17, No. 1, January 1, 1971, pp. 32-38.

Lockett, J.A., A.R. White, "Controlled Tests for Performance Evaluation," The Rand Corp., P-5028, June, 1973.

Mayo, E., THE HUMAN PROBLEMS OF AN INDUSTRIAL CIVILIZATION, Macmillan, New York, 1933.

Roethlisberger, F.J., and W.J. Dickson, MANAGEMENT AND THE WORKER, Cambridge, Mass., Harvard University Press, 1939.

Seven, M.J., B.W. Boehm, and R.A. Watson, "A Study of User Behavior in Problem Solving with an Interactive Computer," The Rand Corp., R-513-NASA, April, 1971.

Sharpe, William F., ECONOMICS OF COMPUTING, Columbia University Press, 1969.

Watson, R.A., "The Use of Computer System Accounting Data to Measure the Effects of a System Modification," The Rand Corp., P-4536-1, March, 1971.

DOLLAR EFFECTIVENESS EVALUATION OF COMPUTING SYSTEMS

Leo J. Cohen

Performance Development Corporation

ABSTRACT

Our technical evaluations of computing system performance tend to concentrate on specific aspects of capability or capacity. The dimensions for such measurements that are involved are scientific in nature; CPU-seconds, byte seconds, and so on. Generally speaking, these dimensions are incompatible and must be conjoined via analysis into a synthezised view of the whole. Of particular interest to this paper, is the problem of creating a dollar evaluation that is related to the configuration as a capacity, and to the use of that capacity by loads in all equipment categories.

The approach adopted is the development of the concept of dollar use of the various configuration equipments relative to the dollar capacity that these equipments make available. The use dollars are developed from observed measures of technical performance, and the method conjoins these various measures into a set of dollar related measures of overall performance.

The measures of overall performance in dollar value terms leads naturally to a set of figures of merit that are referred to as dollar effectiveness. It is then shown, with a 3-system example, that these dollar effectiveness measures provide absolute measures of comparability between different systems in terms of the value of technical performance relative to total dollars of cost.

INTRODUCTION

In dealing with the CPU, the measurement dimension is often the CPU-second, CPU-minute, and so on, depending on the load unit that is most convenient. For the peripheral equipments, we measure data transfer load in terms of device-minutes, etc., while for the memory the dimensions are usually of the form word-second, or byte second. That is, the dimensions chosen for measurements made in each of the equipment categories best serves those categories without overt concern for their compatibility. This means that there is no direct approach available, based on these dimensions, that will allow us to unify the performance figures for each of the equipment categories into a homogeneous measurement that is capable of providing a scale of performance suited to comparative evaluation. Therefore, we seek a new dimension which subsumes performance measurements in all equipment categories and conjoins them into a single measurement value that can then be compared. As a consequence, on such an "absolute" scale of performance the relative capabilities of two systems can be evaluated. To accomplish this the dimension required is "dollars". Thus we cast dollars - the universal measure of material value - in the role of performance measurement unit for the entire system. In the broadest view we will say that the "dollar capacity" of our system is its total cost per hour, per month, or as depreciated, and we will distribute that capacity into "use" categories. We will then measure the useful work obtained from these dollars against the totality of dollars involved.

Use Categories

The three major categories of a computer configuration are the CPU, the memory, and the peripheral equipments. For the CPU there are four load categories into which we can distribute the total CPU capacity. These categories are application operating system, cycled and idled CPU load. Thus, in a particular instance of CPU utilization there will be, in each of these categories, a certain load representing a portion of this capacity. We define this load, determined after the fact, as a "use". For example, suppose that in a period of observation of sixty seconds, 25 seconds was given over to the execution of application programs. We will then say that the application use of the CPU was 25 seconds. Similarly, there will be an operating system use of the CPU, and so on, for each of the four CPU categories.

Memory use will be defined in an analgous fashion. If 120 kilobyte seconds of memory load are observed for, say, application programs over a given performance period, then the application use of the memory was 120 KBS. Thus, the notion of use is after the fact, whereas the concept of load appears as a specification of a requirement on the system given in advance. This is precisely the context in which the notion of equipment use should be understood. That is, we seek to determine ways in which the dollars of the system are used, and the implication is that the determination can only be made after an observation of the system has occurred.

Each of these equipments can be associated with a capacity and the loads imposed on them can be distributed over the four use categories described above. The discussion of dollar effectiveness that follows is dependent on this distribution of loads by equipment type and usage, and in the appendix a detailed accounting of the usage categories is made for the various equipments. However, the discussion of the dollar effectiveness techniques should provide a suitable contextual definition of these elements, and it is therefore recommended that the reader reserve his review of their formal specifications.

DOLLAR USE

The equipments of the computer configuration can be associated with a cost per unit of time. Since the load for each of these elements is already distributed over the four load categories, it follows that an equitable distribution of the individual costs would be on the basis of proportional load in each category. Thus, that proportion of the total CPU load which is due to execution of application programs will be said to account for a proportionate amount of the cost for the CPU itself. For example, suppose that 50% of the total CPU load is due to applications, 20% due to the operating system, 5% due to cycling, and 25% to idling. If the total cost for the CPU were $90 per hour, then this cost would be distributed into four "dollar use", categories; application use, operating system use, cycle use, and idle use. In the present example, the $90 would therefore be distributed into $45 of application use, $18 of operating system use, $4.50 of cycle use, and $22.50 of idle use. It should be noted that just as the four load categories are all-inclusive and mutually exclusive, so also are the four use categories. This means that these use categories must have their individual dollar values summing up to the total cost per unit time for the equipment.

It should be obvious that the technique described above can be extended to all of the equipments of the configuration. This follows directly from the fact that for each type of equipment a definition of each of the four load categories has been established. Dividing the load observed by the period of the observation results in the load rate, and it is this rate that then gives rise to associated dollar use for the equipment.

The Distribution Matrix

Figure 1 is an example of a distribution matrix. In the left hand column of the matrix the various equipments of the configuration have been listed. In this example, the configuration consists of the CPU and memory, two channels, a disk, and three tapes. For the sake of simplicity the controllers for the devices have been excluded from the example.

The next column of Figure 1 gives the total cost per hour for each equipment listed in column one. This is then followed by four column pairs, one pair for each of the rate and use categories. The data of the first row thus represents the distribution of the $90 per hour total cost for the CPU into the application, operating system, cycle and idle dollar use categories. The second row shows the distribution of the $110 per hour cost of the memory. Here the distribution matrix indicates that $33 of this total cost was used in the execution of application programs, or equivalently, by users in solving their problems. Of the total cost per hour $2.20 was spent on the operating system, and $30.90 represents the cost of cycling of the total $110 cost per hour for the memory, 40% of the memory was unused, and as a consequence $44 of that expense represents the price paid for this idle capacity.

Looking at the two rows associated with channels one and two, it can be seen that their individual cost per hour is $25. Channel one evidently connects both application and operating system files to the core memory since there are rates shown in both of these categories. As a result, there is an operating system use as well as an application use for the employment of channel one. These figures are 75¢ and $4.50 respectively. On the other hand, channel two apparently connects no operating system files to the core memory, and therefore the operating system rate and associated operating system dollar use for this channel are both zero. The figures indicated in the row for the disk show that the operating system and application files are both resident on that device, whereas the tape equipments all indicate that only application programs make reference to tape files.

Overall, each equipment of the configuration is associated with a total cost, a portion of which is attributed to application use. In one sense, the best possible system is that one in which the user, through applications programs, derived a maximum dollar benefit. To measure the degree of dollar benefit derived by users from the system, all dollar use values in the application category are added

	Total Cost per Hour	Appl. Rate	Appl. $ use	O/S Rate	O/S $ use	Cycle Rate	Cycle $ use	Idle Rate	Idle $ use
CPU	90	.60	54.00	.20	18.00	.05	4.50	.15	13.50
MEMORY	110	.30	33.00	.02	2.20	.28	30.80	.40	44.00
CHANNEL 1	25	.18	4.50	.03	.75	.54	13.50	.25	6.25
CHANNEL 2	25	.26	6.50	0	0	.34	8.50	.40	10.00
DISK	40	.18	7.20	.03	1.20	.54	21.00	.25	10.00
TAPE 1	40	.11	4.40	0	0	.15	6.00	.74	29.60
TAPE 2	40	.09	3.60	0	0	.11	4.40	.80	32.00
TAPE 3	40	.06	2.40	0	0	.08	3.20	.86	34.40
TOTALS	$410		$115.60		$22.15		$92.50		$179.75

THE DISTRIBUTION MATRIX

FIGURE 1

together. In the example this is $115.60. Thus, of the total $410 per hour cost for the system, application jobs use these dollars to the extent of $115.60 for problem solving. Adding up the operating system dollar use values we see that $22.15 of the $410 cost per hour is necessary to produce the results of application program execution. Furthermore, the example shows that a total of $92.50 was expended in cycling the various facilities of the system. This must be accounted as a cost for producing user results as well. Finally, the total cost of the idle facilities is $179.75.

A CONFIGURATION EXAMPLE

As mentioned earlier, the concept of dollar effectiveness is to be applied system wide. In order to derive measures that conjoin the effects of each of the equipment categories over each of the loading categories, we demonstrate the application of the dollar effectiveness technique to a system configuration problem.

We will first consider a system having the "Model A" mainframe, with two channels, a disk set consisting of a number of disk packs on a common controller, and two tapes. The configuration is shown in Figure 2. It also includes a multiplexor connecting a printer and a card reader. As the figure shows, both the application programs and the operating system have files on the disk set, and therefore there will be a degree of conflict between the application jobs and the operating system for references to the disk set. On the other hand, channel 2 connects the two tapes through the common controller and thereby makes application program files available. In particular it should be noted that the operating system has no files on these tapes and therefore makes no reference to them.

For the second configuration the same mainframe will be employed but an additional disk set and channel will be added. These additional equipments will be used for files associated with the application programs only. In the third configuration these two channels and associated disk sets are retained, but the mainframe is changed from a Model A to the slower "Model B".

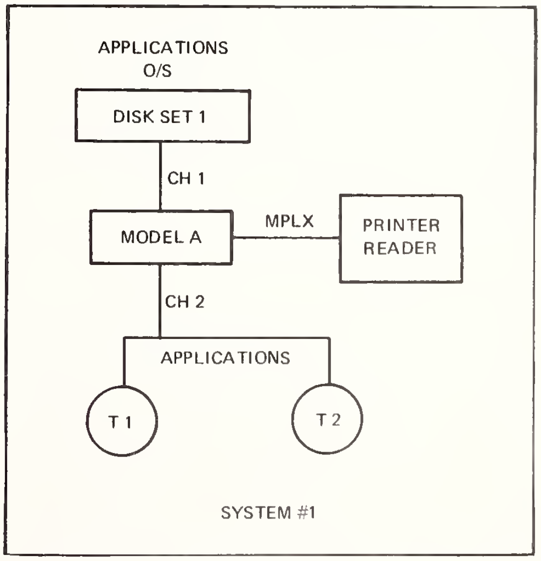


FIGURE 2

We will consider the distribution matrix for each of these configurations. However, for the sake of simplicity we have excluded all details of controller performance, assuming instead that since there is one controller per channel, the controller performance is sufficiently represented as part of the channel cost. The figures for the multiplexor and its attached devices have also been left out of the examples. This is done in the interests of simplicity and economy of discussion. However, in a formal analysis, all equipment categories should be represented, particularly where their individual costs may be a significant fraction of the whole. Finally, the several disk packs of a unit have been brought together under a single heading and designated as a disk set. That is, in the discussions that follow they are to be treated as a composite entity. A more accurate analysis would, of course, separate these, but this simplifying representation is sufficient for our purposes here.

Benchmark Loads

If a comparison of performance between systems is to be conducted, a standard load characterizing the installation's activities must be imposed on both of them. Such loads are usually referred to as benchmarks.

Benchmarks are most often in the form of programs. However, in (1) a rather different technique is put forward that involves CPU, memory and data transfer loads. Whichever the method, the basic objective is the same; namely, to establish a standard load that is reasonably descriptive of the way in which the computer is employed.

It is just such a benchmark load that is required for a dollar effectiveness evaluation. This benchmark will be the same for all systems studied and when imposed will create loads in all of the equipment categories.

System 1

The first configuration, shown in Figure 2, is referred to as System 1, and Figure 3 is the distribution matrix for this system. In this figure all equipments of the configuration with the exception of the controllers, multiplexor, printer and card reader are listed in the left hand column. The next column lists all of the costs for these equipments, showing them as monthly figures. Thus, the CPU cost is $6740.00 per month, the memory $10,060.00 per month, the channels $700.00 per month and so on. The total monthly cost is $21,964.00.

The next four column pairs represent each of the four use categories for these equipments. For example, the rate at which the CPU was employed for application jobs is .432. Therefore, of the total $6,740.00 cost for the CPU, application jobs have gotten .432*6740=2912 dollars of use per month.

With the operating system rate (O/S) at .212, the operating system use of the total CPU dollars is $1429.00. In the last two column pairs, we see that the cycle use and idle use of the CPU, in dollar terms, are $613.00 and $1786.00 respectively.

The distribution for $10,060.00 per month cost of the memory is along the same lines, based on the rate figures shown. It might be noted that the operating system memory use is particularly high, suggesting that the operating system is of extreme size. The next largest of these is for cycle use, followed by the application and idle use, in that order. The size of the cycle figure in fact, suggests that users are requesting larger blocks of storage than they can employ.

Consider now the figures for channel 1 and disk set 1. In the application category the rate at which application jobs employed the disk was .297. However, channel 1 was employed at a rate of .231. The disparity in these rates can be accounted for in terms of seek operations on the disk packs. That is, during seek activity there were periods when no data was being transferred over the channel and therefore the application rate for the channel in that period is zero. It should be noted that the operating system rate for disk set 1 is .126 and that there is also a reduction in the operating system rate for channel 1 as well. Now consider the cycle rate for disk set 1. This indicates that 31.2% of the time, files on this disk set were active but unused. Looking at the cycle rate for channel 1, the figure .382 represents an increase over the cycle rate for the disk set that is equal to the decrease in the channel rates for application and operating system together. Thus, the decreased utilization of the channel in data transfer activities has turned up as increased cycle rate.

It should be noticed that the idle rate for channel 1 and its disk set is equal to the idle rate for the CPU. The implication is that whenever an application job was present in the system the programs executed by that job had active files on the disk set. This is not the case with tapes 1 and 2, however, since

	Cost	Application		OVERHEAD				Idle	
				O/S		Cycle			
		Rate	Use	Rate	Use	Rate	Use	Rate	Use
CPU	6740	.432	2912	.212	1429	.091	613	.265	1786
MEMORY	10060	.248	2495	.375	3772	.261	2626	.116	1167
CHANNEL 1	700	.231	162	.120	84	.384	269	.265	185
CHANNEL 2	700	.205	144	0	0	.203	142	.592	414
DISK SET 1	2624	.297	779	.126	331	.312	819	.265	695
TAPE 1	570	.091	52	0	0	.076	44	.833	474
TAPE 2	570	.114	65	0	0	.127	73	.759	432
TOTALS	21964		6609		5616		4586		5153

DISTRIBUTION MATRIX

SYSTEM 1

FIGURE 3

their idle rates are both a good deal higher than the idle CPU rate. This means, of course, that some jobs did not execute application programs having active files on the tapes. On the other hand, the idle rate for channel 2 is .592 and this is significantly lower than the idle rates for either of these tapes. The reason for this becomes clear if we consider the application rates for the channel and the two tapes. The channel 2 application rate is .205 and is the sum of the two application rates for the tapes. Thus, the tapes were necessarily utilized sequentially since there is only a single channel. However, the cycle rate for channel 2 is .203, and this is the sum of the cycle rates for these two tapes. This means that files were active but unused on these tapes "sequentially"; i.e., at no point did any job execute requiring the files on both tapes, and furthermore at no time were there two tape jobs executing concurrently. Thus, the cycle rate for channel 2 is maximum while its idle rate is minimized.

Measures of Effectiveness

By summing each of the dollar use columns in the distribution matrix for System 1, we arrive at total dollar use figures in each of the four categories. Thus, of the total $21,964.00 per month cost for the system, $6609.00 of application use is obtained. Operating system use is $5616.00, and the cycle is $4586.00. These two together represent an overhead dollar use of $10,202.00. Finally, out of the $21,964.00 of monthly cost, a total of $5153.00 is idle.

Now consider the ratio of the total application use of $6609.00 to the total monthly cost of $21,964.00. Expressed as a percentage, we have that 30.0% of the total cost was used for application job execution. On the other hand, the $10,202.00 of overhead cost, combining operating system and cycle use dollars, represents a 46.4% use of the total monthly dollars expended. And, by similar calculation, 23.5% of the total dollars of system cost idle.

We will refer to the ratio of application use to total cost as the system's "dollar effectiveness". The ratio of the sum of operating system and cycle dollars to the total cost will be referred to as the "effectiveness cost", and the ratio of the total idle dollars to total cost as the "excess capacity cost". We will develop from these effectiveness figures a basis for comparison of systems, or configurations of a given system. It is toward this objective that we are working in the sections to follow.

System 2

The configuration for System 2 augments that of System 1 by the addition of a second disk set and an associated channel. Figure 4 shows this configuration. Note that the mainframe remains a Model A, and that disk set 1 is now devoted entirely to the operating system files, while all application program files have been moved to disk set 2. The new channel is referred to as channel 3, and the configuration retains channel 2 and attached tapes, as well as the multiplexor and printer/reader combination.

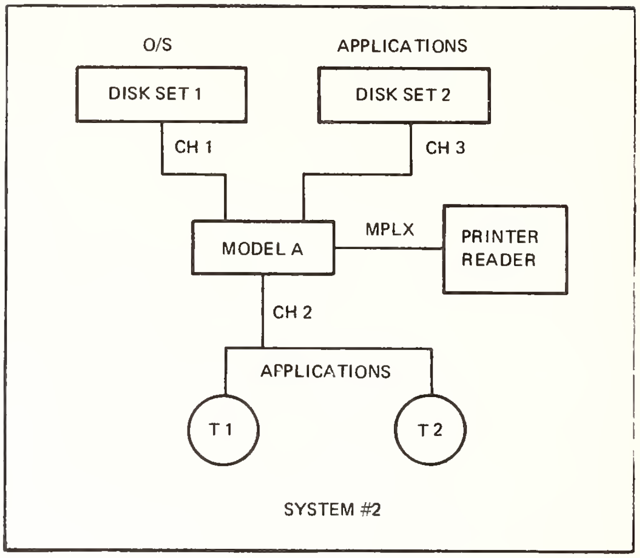


FIGURE 4

The distribution matrix for System 2 is shown in Figure 5. The equipment list in the left hand column has been modified to include the new channel and disk set along with the additional cost of these requirements. More specifically, the equipment costs have increased by the $700.00 for the third channel and the $2624.00 for the disk set 2, to a total of $25,288.00. But this column of figures is not the only place where there is an effect in the distribution matrix. The addition of data transfer capability will also have a general effect that will be reflected in the dollar effectiveness figures.

The first observation is with regard to the cycle rate for the CPU. With the installation of the additional peripheral devices there is a reduction in data transfer conflicts between the operating system and the application jobs, and this results in a lower cycle rate. As the figures show, the cycle rate goes from .091 down to .087. However, since the test is made while executing the same set of jobs as used in System 1, and since the period of observation is kept constant, it is clear that the CPU rates for both the application and operating system jobs in System 2 must be the same as that in System 1. Therefore, the decrease in the cycle rate from System 1 to System 2 appears entirely as an increase in idle rate in System 2.

The effect on the cycle rate of the equipment reconfiguration is rather small because the application jobs executed for these experiments do not create a high volume of data transfer. In fact the positive effect of lower cycle rate for the CPU is lost to higher cycle rates and associated cycle dollars for the peripheral equipment. This can be seen by looking at Figure 5 for disk set 1. The effect of the reconfiguration has been to make the operating system files available to the operating system jobs without interference from application jobs. On the other hand, the only activity on this disk set is with respect to operating system files in order to operate the application jobs. Thus disk set 1 is idle only when there are no application jobs in the system, and therefore the idle rate is .269. Note that the data transfer load for this disk set remains at .126, representing the data transfer load created by the operating system in making reference to its files. Since there is no application job reference to this disk set in the configuration, it follows that the remaining load appears in the cycle category, and as the distribution matrix shows, the cycle rate for this disk set is .605.

Now consider disk set 2. This disk set is devoted entirely to the files for programs executed by application jobs, and therefore the data transfer load created on this disk set will be the same as in System 1 since these are the same jobs. The rate for the application jobs is therefore .297. However, the absence of operating system data transfer load on this disk set means that the remainder of this capacity must be distributed between cycle and idle loads. For this configuration, the cycle rate for disk set is .481. Thus we see that the reduction in cycle rate of the CPU, as a consequence of the addition of the new disk set and

	Cost	Application		OVERHEAD				Idle	
				O/S		Cycle			
		Rate	Use	Rate	Use	Rate	Use	Rate	Use
CPU	6740	.432	2912	.212	1429	.087	586	.269	1813
MEMORY	10060	.246	2474	.375	3771	.259	2605	.120	1210
CHANNEL 1	700	0	0	.121	85	.610	427	.269	188
CHANNEL 2	700	.205	144	0	0	.203	142	.592	414
CHANNEL 3	700	.233	163	0	0	.222	155	.546	382
DISK SET 1	2624	0	0	.126	331	.605	1588	.269	705
DISK SET 2	2624	.297	779	0	0	.222	583	.481	1262
TAPE 1	570	.091	52	0	0	.076	43	.833	475
TAPE 2	570	.114	65	0	0	.127	72	.759	433
TOTALS	25288		6589		5616		6201		6882

DISTRIBUTION MATRIX

SYSTEM 2

FIGURE 5

attendant channels, must be associated with some increases in both the cycle and idle rates of these equipments, and as the figures of the distribution matrix show, with significant additional costs in these categories.

A similar situation applies to the channels of this configuration. As the distribution matrix shows, there is no longer an application rate for channel 1, while there is a .121 operating system rate, and an idle rate of .269. This means that the remaining capacity for the channel is concentrated in its cycle rate, and that is .610. It should be noted that the idle rate for the channel is equal to the idle rate for the CPU. This channel is associated with the operating system files which are active whenever there is an application job in the system.

Channel 3, the new channel of the configuration, is associated with disk set 2 and the files of the application jobs. We see that the application rate is .233 for this channel, with the difference between this rate and that for disk set 2 being attributed to seeking activity on the disk in which the channel is not involved. The cycle rate for this channel is .222 and its idle rate is .546. Thus, the reconfiguration has added dollars in the cycle and idle categories for the channels as well.

The overall effect of the reconfiguration can be appreciated in terms of the dollar effectiveness measures. By summing each of the dollar use columns in our four categories and making the effectiveness calculations that were applied earlier to System 1, we find that the dollar effectiveness is 26.1% and that the effectiveness cost is 46.7%. This is derived from the sum of the operating system and cycle use figures, and is significantly different for System 2 simply because the improvement in the mainframe cycle rates and associated dollars is more than offset by the increase in the cost of cycled peripheral equipment. The indication is, of course, that too much of the increased dollar capacity of the system has gone into overhead use, while the application use of these dollars has remained almost the same. To put it differently, the dollar effectiveness has dropped while the effectiveness cost has increased. Finally, our calculations also show that a significant portion of the increased cost of the system has gone into idle dollars, since the cost of the excess capacity is computed to be 27.2%.

System 3

It was stated earlier that the amount of CPU activity relative to the data transfer activity is high. Yet, as the distribution matrix figures show, there is a significant amount of idled CPU load remaining in the period of observation. Let us suppose that our long range interests are potentially satisfied by an excess storage capability represented by the configuration of System 2. On the other hand, if our processing load at the CPU is not expected to rise appreciably, it is reasonable to consider changing the configuration of System 2 to the extent that it involves a slower but less expensive CPU. We refer to this configuration, shown in Figure 6, as System 3. The slower CPU is referred to as the "Model B", but the remainder of the configuration is unchanged from that of System 2. That is, we retain the two disk sets and the tapes with their associated channels. The application and operating system files are separated on the disk sets as before.

The major property of System 3 is, of course, the change in the CPU from a Model A to a Model B. There will consequently be a number of effects since the CPU for Model B is approximately one half as fast as the one for Model A, and its cost is approximately 1/3 less. This gives a total cost of $23,368.00 as compared to $25,288.00 for System 2.

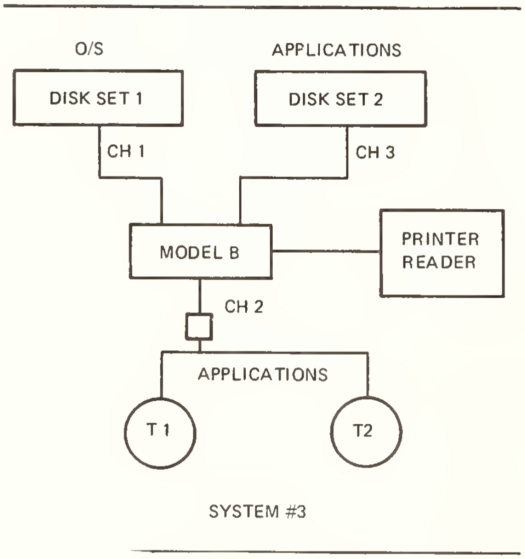


FIGURE 6

Both the rate and the dollar effects are immediately apparent in the first line of Figure 7, showing the distributions for the CPU. As can be seen, the total monthly cost for the CPU is $4,820.00. The application rate has jumped to .603, the operating system rate to .296, while the cycle rate has fallen to .013. The idle rate, representing the remainder capacity for the CPU, is nearly at zero.

As a consequence of the slower CPU there will be an extension in the execution time of applications jobs, so that there will also be a corresponding increase in the memory loads associated with the application jobs. Thus, the memory application rate is .270, the operating system rate .412 and the cycle rate .284. These increases have been derived from the remainder capacity of the memory so that the memory idle rate is at .034.

The data transfer loads and associated rates for the application jobs have remained the same in this configuration. This is also true for the operating system data transfer load relative to disk set 1 and the associated channel. However, because the application jobs are present in the system for a greater execution time, the files associated with these jobs are active for a correspondingly greater time period. Because the application load has remained the same, however, this implies an increase in the cycled load and associated cycle rates. The distribution matrix indicates that these cycle rate increases are significant.

Again, each of the dollar use figures are totaled at the bottom of the distribution matrix. This shows the dollars of application use at $6,825.00 while the total overhead dollars are $13,968.00, and the total idle dollars are $2,575.00. Our effectiveness calculations then show that the dollar effectiveness is 29.2%, the effectiveness cost 59.8%, and the idle cost 11.0%.

ABSOLUTE EFFECTIVENESS

The sense in which we have been pursuing the concept of dollar effectiveness is related to the degree of effective use, or application, that is derived from the total dollars spent on the system. Thus, the dollar effectiveness figure that was defined earlier is a measure of the percentage of these total dollars that are effective in problem solving. But to compare the performance of two systems on a dollar basis requires that we take into account not only the effectiveness of the system's cost, but also the specific cost to achieve that effectiveness. That is, a high dollar effectiveness at a high effectiveness cost implies a low excess capacity cost. If the two systems under consideration have equal total cost, then clearly the higher the dollar effectiveness and lower the effectiveness cost, the better the effectiveness per unit cost. We can formalize this expression by dividing the dollar effectiveness figure by the effectiveness cost figure for each of the computing

	Cost	Application		OVERHEAD				Idle	
				O/S		Cycle			
		Rate	Use	Rate	Use	Rate	Use	Rate	Use
CPU	4820	.603	2906	.296	1427	.013	63	.088	424
MEMORY	10060	.270	2716	.412	4145	.284	2857	.034	342
CHANNEL 1	700	0	0	.121	85	.791	553	.088	62
CHANNEL 2	700	.205	144	0	0	.616	431	.179	125
CHANNEL 3	700	.233	163	0	0	.597	418	.170	119
DISK SET 1	2624	0	0	.126	331	.786	2062	.088	231
DISK SET 2	2624	.297	779	0	0	.552	1449	.151	396
TAPE 1	570	.091	52	0	0	.096	54	.813	464
TAPE 2	570	.114	65	0	0	.165	93	.721	412
TOTALS	23368		6825		5988		7980		2575

DISTRIBUTION MATRIX

SYSTEM 3

FIGURE 7

systems under consideration. We can then normalize their different total costs to derive a measure that shall be referred to as "absolute effectiveness", which is useful in the comparative evaluation of systems.

Sample Comparisons

To demonstrate the absolute effectiveness idea, we have listed the three effectiveness computations for each of the three systems in Figure 8. This shows dollar effectiveness, effectiveness cost, and excess capacity cost foreach of the systems and with these figures, several other figures that are derived from them. Let us first, however, recapitulate some of the evaluation parameters that we established in the development of this configuration problem.

We began with a given load of jobs which was seen to be relatively "compute bound", and with a significant amount of idled load available in our first system, designated as System 1. We then adjusted the configuration to System 2 and stated that an excess data transfer capability was desirable for future requirements. It was presumed that the CPU requirement would not appreciably increase, and we considered a lower cost configuration having a slower CPU that was designated as System 3. For the scope of the present problem, we will consider the requirement for increased data transfer capability, used to justify the configuration of System 2 as being of critical interest, so that we now seek to evaluate the relative merits of System 2 and System 3 in dollar effectiveness terms.

Note first that the dollar effectiveness for these two systems is 26.1% and 29.2% respectively, while their respective effectiveness costs are 46.7% and 59.8%. By inspection of these figures one can see that the improvement in dollar effectiveness in System 3 is more than off-set by a coincident increase in effectiveness cost. For System 2 this means that for every dollar spent to run the system (overhead), 55.8 cents were employed in problem solving. On the other hand, for System 3, only 48.8 cents of each dollar was employed in problem solving. To put this another way, the ratio of dollar effectiveness to effectiveness cost is best when maximum. This occurs when the dollar effectiveness is maximum and the effectiveness cost is minimum. Hence the effectiveness per unit cost represents a measure of application use relative to the cost of system operation. However, effectiveness per unit cost does not take into account the total cost of the system. That is, the dollar effectiveness and effectiveness cost values are both derived relative to total system cost, so that their ratio removes this cost factor. If, therefore, the effectiveness per unit cost is divided by the total cost, we will get a figure that is referred to as "unit effectiveness per dollar", or more simply as "unit effectiveness". This unit effectiveness is

	System #1	System #2	System #3
DOLLAR EFFECTIVENESS (A)	30.0%	26.1%	29.2%
EFFECTIVENESS COST (B)	46.4%	46.7%	59.8%
EXCESS CAPACITY COST	23.5%	27.2%	11.0%
EFFECTIVENESS/UNIT COST (C=A/B)	.646	.558	.488
TOTAL COST (D)	$21,964	$25,288	$23,368
EFFECTIVENESS/DOLLAR (10,00 C/D)	.294	.220	.208
ABSOLUTE EFFECTIVENESS			

FIGURE 8

actually the dollar effectiveness divided by the product of the effectiveness cost and the total cost. Thus the higher the dollar effectiveness and the lower the effectiveness and total costs, the greater will the unit effectiveness figure be.

More broadly, if unit effectiveness represents the number of dollars of use relative to the dollars of cost, then unit effectiveness measures this on a per dollar basis, and as such it is a figure of merit that is directly comparable for two systems of differing total cost.

In the table of Figure 8, effectiveness per unit cost for System 2 is .558 and for System 3, .488. Dividing these by their respective total costs and multiplying the result by 10,000 to place the dividend on a convenient scale, gives the unit effectiveness value 220 for System 2, and .208 for System 3. Thus, the cheaper system does not provide as much performance for the dollar as the more expensive one. However, we may not interpret this as meaning that for accomplishing the same job as represented by the CPU and the data transfer loads exerted by application jobs on the two systems, the performance of System 2 relative to its total cost of $25,288.00 is better than that for System 3 at its total cost of $23,368.00. That is, the total number of dollars that are unused, and which is referred to in Figure 8 as the excess capacity cost, is significantly higher in System 2 than in System 3. This means that in System 2, 27.2% of the total cost of the system is unused, and represents dollars that are available for employment. Of course not all of these dollars can be turned into application dollars. However, the greater this excess cost figure is, the greater the potential for conversion into application dollars. This implies then, that for System 3 the 11.0% excess capacity cost figure leaves little room for such conversion. Furthermore, since the CPU capacity of System 3 is almost entirely employed by the current job load, employment of the available data transfer capacity, represented by cycled data transfer loads will have to be realized by increasing the data transfer loads created by the current jobs. This means changing the nature of these jobs, which provided one of the parameters of our evaluation problem as stated earlier. This would not be the case for System 2, however, since the excess CPU capacity implies that we might run our current jobs, plus new jobs that create additional data transfer load.

Lost Dollars

In order to be able to employ idled peripheral equipment load we must have some idled CPU load available. That is, to use the peripheral equipments it is necessary to first execute instructions on the mainframe. Therefore what of the case where the rate of utilization of the CPU leaves the idled CPU load at or near zero? This is effectively the circumstance in Figure 7 where the idled CPU rate is only 8.8%. For all intents and purposes we may say in this situation that there is virtually no more CPU "room" for the execution of additional jobs, and as a consequence the various idled rates for the peripheral equipment represent wasted assets and their associated idled dollars are wasted as well. We will refer to these as "lost dollars".

Of course the same may be said for the memory loads that cannot be utilized because there is no more available CPU capacity. This is hardly a question in the example of Figure 7 since the idled memory rate is only 3.4% and this suggests that even if there were an available idled CPU rate there is no room in the memory for the execution of any further jobs.

This question of lost dollars can be given a somewhat broader scope if one raises the argument that even though idled load isn't available at the CPU for the introduction of additional jobs, there may be a significant cycle load available. That is, by increasing the average degree of multiprogramming we might indeed be able to have its CPU load requirements satisfied by the available cycled CPU load. Of course for this to have any effectiveness whatsoever it is necessary that the new jobs introduced be CPU only, or highly CPU bound. Such jobs do not use the peripheral equipments and as a consequence we will still have our idled load and again the same lost dollars.

In summary then, once the idled CPU rate gets particularly low (and/or the idled memory rate) we can expect that idled capacity in the peripheral equipment translates into lost dollars.

The significance of lost dollars is that they represent money paid for the configuration which can never be utilized. If tuning has been accomplished so that both the main frame and peripheral equipments are operating optimally for the loads we impose, then it follows that little short of a juggling of the job schedule can do much to recover any of these lost dollars. In fact, it is most likely the case that modification of a current schedule that produces satisfactory thruput and turnaround will modify these performance figures of merit as well, so that in exchange for fewer lost dollars the installation pays the price of lesser technical performance.

CLOSING REMARK

The dollar effectiveness approach to performance analysis is designed to give performance management a means by which the total effectiveness of the system over all equipment categories - CPU, memory, channel, controllers, and peripheral devices can be measured. It furthermore can be thought of as an overview approach to technical performance in that it relates performance in all of these equipment categories in terms of distribution of the various loads imposed on the system by applications, and the loads that occur as a result. This therefore provides an opportunity for performance management to gauge the value of future needs; i.e., growth capability, and to take this into account when planning the acquisition of new equipment to modify the current system, or a complete system to replace it.

APPENDIX

In this section we turn our attention to a more detailed investigation of the data transfer characteristics of the equipment in the data transfer chain. These equipments will include the channels, the controllers, and the devices. The objective is to define categories of use for these equipments so that these categories are compatible with the application, operating system, cycled and idled categories defined for the CPU and memory.

Channels and Active Files

The channel is considered as a necessary connector between file and core memory. This implies a connection to a particular device or set of devices. We define the channel as being idle when either one of two conditions exist. The first of these is that there are no jobs at all in the system; the second condition is that no job currently present in the system executes a program that makes reference to any file on any device connected by the channel. The condition for an idle channel is of course obvious when the device has a removable media, such as a magnetic tape or disk pack. In such cases, when no media is mounted, the device clearly has no file present that will be referenced by a program execution. For devices with permanently mounted media, however, such as a drum, the files carried by the device are permanently available. In this case, the criteria is whether or not that availability is of interest to any job currently executing in the system. When some job in the system executes a program making reference to a given file, we assume that a data transfer reference to that file is potentially likely and that the job could not run without the presence of the file on the device. By extension, then, the job could not run without the presence of the channel for communicating the data of that file to the core memory. Thus the channel is deemed necessary when a file associated with a given job execution is present on the associated peripheral device, and unnecessary otherwise. It is the time period over which the latter condition exists that is referred to as "idled channel" time, and we will call such files "inactive". On the other hand, files associated with current jobs will be called "active".

Consider now the case where the channel connects core memory to a device holding a file referenced by one of the jobs currently executing in the system; under these conditions the channel is not idled. To distinguish between a channel's active time, that is the time it is transferring data, and its inactive but non-idled time, we shall refer to the latter as "cycled channel" time. The non-idled, non-cycled time for the channel can then be distributed into two categories. The first of these is the "application channel" time, which applies whenever data is transferred between core memory and the device for an application job. The second category is "operating system channel" time, and is applied whenever the data transferred across the channel is associated with an operating system file.

Thus, we have distributed the total time of channel availability into four categories: When the channel is transmitting data associated with an application file, application channel time is accumulated; when a file associated with the operating system is employing the channel to transfer data, operating system channel time is accumulated; if no jobs are present in the system or no file on the device connected by the channel is associated with any job currently in the system, the channel is idled; all time remaining in the period of observation is therefore cycled channel time.

Suppose now that the channel is associated with not just one device, but with a set of devices. Such a situation occurs when the channel leads to a set of disk packs or a set of magnetic tapes. It is then sufficient for the purpose of channel load definition to consider all of the devices that are accessible by the channel as a single device. That is, if no device of the group is associated with any job currently active in the system, we will also say that the channel is idled.

Multiple Channels

The same definition of idled channel load also applies to the case where more than one channel is associated with a common group of devices. Thus, suppose that with a suitable controller design, two channels could connect to the same group of disk packs. Then both channels are idle whenever no disk pack is mounted or when no files on any of the mounted disk packs pertain to any jobs currently operating in the system. This parallel configuration, however, does present some minor difficulty in the definition of cycled channel load. That is, suppose that the level of activity in the system is sufficiently low so that only one of the two channels to the set of devices it is also inactive. Therefore, we must define this unused channel capacity as cycled load. The argument here is that the definition of cycled load is independent of the degree of activity at a device, or relative to the files held by the device. In fact, if there were a second, separated group of devices, with only a single channel connection and a much higher activity, the meaning of cycled load on the unused channel relative to the device group with very low activity is reasonably clear. That is, in this case the allocated but unused channel facility is not only wasted but misplaced, and designation of this unused facility time as cycled load rather than idled load underlines this situation.

Floating Channels

A possible configuration for data transfer channels in a system is as "floating" channels. This means that any channel not currently involved in a data transfer activity may be assigned to any specified device in the configuration. Under such circumstances all channels are creating idled load when there are no jobs in the system or no files mounted on any devices reachable by the floating channels. Suppose now that there are C floating channels, and that there are F active files in the system. If the number of active files is less than the number of floating channels, we will say that C - F channels are idled. Otherwise, if the number of active files is greater than or equal to the number of floating channels, no channels will be inactive and no idled channel load is created. Figure 9 depicts floating channels under different usage conditions.

Rather than define cycled load directly in an environment involving floating channels, it is better to compute the cycled load for all channels as the remainder after the application, operating system, and idled loads have been accounted for. Thus we can think of channel capacity in terms of, say, so many "channel-seconds". The distribution of this capacity must then be over each of the four load categories, leaving one of them, in this case cycled load, to be calculated from the rest.

Multiplexors

Multiplexor sub-channels operating in a non-burst mode can be considered as a collection of individual floating channels relative to the devices that they connect. The distribution of the load on the multiplexor channel is then proportional to the utilization of its sub-channels in the four load categories. When the multiplexor operates in a burst mode, however, it acts as if it were a selector channel. That is, for the period of burst mode operation the entire set of sub-channels of the multiplexor is con-

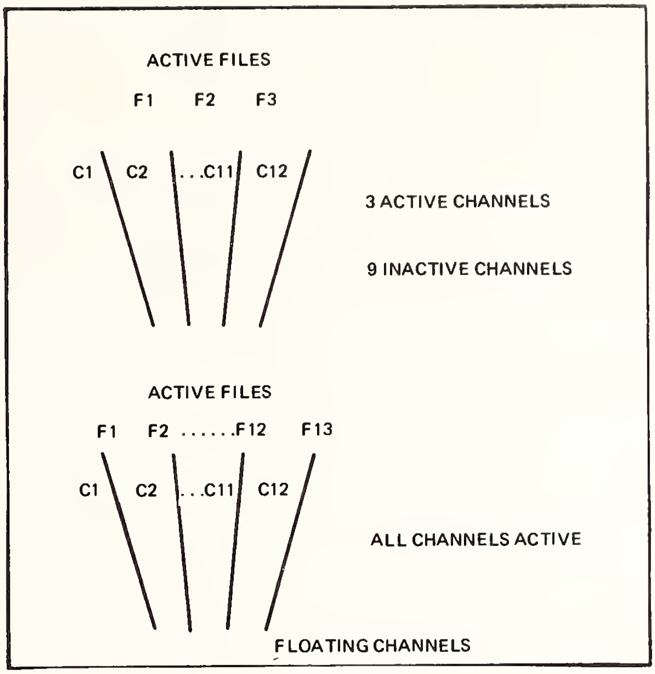


FIGURE 9

nected to, or are available for, data transfer to one specific device. All sub-channels, therefore, accrue either application or operating system load. Otherwise the cycled and idled loads accumulated by the multiplexor are the same as those defined for its non-burst operation.

The Controllers

A controller for a single device or group of devices that is connected to a single selector channel defines its idled and cycled loads in exactly the same way as the channel. The application and operating system loads to be associated with a controller are, of course, defined during the periods of their respective activities. Thus, a controller is idled when all devices with which the controller is associated are either unmounted or contain no active files. It is, of course, idled when there are no jobs in the system. The controller is active, but unused, when there are active but currently quiescent files on its devices. In this case the controller is creating cycled load.

It should be noted that the controller and the channels to which it connects might not have identical load distributions. That is, the controller may connect several channels to a group of devices and therefore the sum of the activity over all channels will be associated with the controller, each in its own load category. Furthermore, a controller may be busy throughout the period of certain operations, while the associated channel is cycling. An example is a controller for a disk pack that is busy through the time of arm movement. Over this time however, the attached channels may be inactive and cycling.

The Devices

The distribution of loads for a peripheral device follows the same pattern as that for the other equipments in the data transfer chain between file and core memory. As before, we will say that the device is idled if no media is mounted on it, or if no jobs are currently present in the system. The device is also idled if it holds no active files.

When the device holds active files the load on the device is distributed into application, operating system and cycled categories, according to the same criteria applied to the channels and controllers.

REFERENCE

Leo J. Cohen, System Performance Measurement and Analysis, Performance Development Corp., Trenton, N.J., May, 1973.

COMPUTER SCHEDULING IN AN MVT ENVIRONMENT

Daniel A. Verbois

U.S. Army Materiel Command

INTRODUCTION

Modern day business data processing systems have changed dramatically over the past ten years. With the advent of third-generation machine power, sophisticated data management and operating systems, and with more extensive use being made of teleprocessing, the modern systems are impressive, indeed. But, along with these advances, a problem has appeared that is significantly different from the early days of data processing. That problem is efficient and effective machine scheduling. The difficulty lies in the fact that it is not a simple matter to translate human requirements and human thought processes into complex machine requirements in such a way so as to realize the maximum utilization of expensive machine resources. The problem is most often solved by simply allowing the computer to accept jobs on a first-come, first-serve basis or by utilizing an unsophisticated software job selection procedure. An example of the former is the practice of loading the job queue with the jobs that are to be executed over a given period of time and executing these jobs in a first-in, first-out manner. An example of the latter is utilization of a feature of the IBM 360/Operating System (OS) termed HASP (Houston Automatic Spooling & Priority). This feature is primarily used in those installations that process large quantities of relatively independent jobs; it is unsuitable for long running mutually dependent processes that access integrated files. The intent of the following discussion is not to pursue the pros and cons of the various scheduling methodologies; rather, it is to describe a program which helps greatly to eliminate many of the problems associated with scheduling of the AMC Commodity Command Standard System (CCSS).

AUTOMATED PRODUCTION SCHEDULER DESCRIPTION

The Automated Production Scheduler (APS) is a COBOL program incorporating many of the basic principles of PERT (Program Evaluation and Review Technique). The scheduler applies these principles to descriptions of computer jobs to evaluate the relationship between a variety of activities that compete for resources and time. A computer job in IBM terminology is defined as "a total processing application comprising one or more related processing programs, such as a weekly payroll, a day's business transactions or the reduction of a collection of test date." Therefore, in order to apply the principles of PERT, it is necessary to describe the workload for the computer in terms of computer jobs, i.e., job profiles must be constructed.

Within the APS data base, each job to be scheduled is identified by a name that is compatible with the IBM job control language naming conventions (See Appendix A). For each name, important operational and scheduling parameters are provided. These are: core requirement, run time (calculated, estimated or observed), the earliest time the job can be started and the latest allowable time for job completion, percentage of CPU time, devices required to support the job, predecessor jobs that must be completed prior to the beginning of each job, number of disk tracks that will be required to support transactions into and out of each job, number of disk tracks that will be free at the successful completion of the job and any special conditions that must be considered, such as: restrictions on updating the same file by two programs simultaneously or the physical mounting of a given file before job initiation. The job profile can be established at any level preferred. For jobs that will be processed on computer systems running under an MFT (Multiprogramming with a Fixed number of Tasks) operating system, it is sufficient to define the profile at the job level as opposed to the step level (jobs are comprised of one or more job steps which are defined as that unit of work associated with one processing program). For those jobs that are to be processed on machines which operate under an MVT (Multiprogramming with a variable number of Tasks) operating system, it may be convenient to define the profile at the step level. In either case, the job profiles can be collected in such a manner that one or more jobs may be scheduled by using an indexing technique. For example, suppose a given application, Weekly Billing, comprises five separate jobs. In building the APS data base the five jobs would be individually described, each with a separate name. In addition, the five job profiles could be "collected" under a separate name to facilitate their manipulation in a remote environment.

Jobs may be scheduled for completion in any period up to 24 hours in length by entering the job names (or collection names) into the computer via a terminal (see Appendix B). The APS program will mix and match the job profiles in such a way so as to produce the optimum sequence of execution. Those jobs that cannot be scheduled within the time frame established are indicated on a separate APS report.

APS REPORTS

Several reports are provided by the scheduler. Each report is optional in the sense that, when operating the terminal in interactive mode, the opportunity is provided to print or suppress printing of each report. The basic reports are:

-Status Report (Appendix C). This report is essentially an extract from the APS data base and reflects major parameters assigned to describe each job selected for scheduling. The option is provided to change any parameter for any job on either a temporary (one time usage) or permanent (physical data base change) basis.

-Unscheduled Runs (Appendix D). As previously indicated, all jobs that cannot be scheduled within the time frame or hardware configuration provided will be shown on a separate report.

-Release Sequence (Appendix E). This report lists the optimum sequence of job release along with estimated start time (SETUP) estimated time of completion (REMOVE) and jobs that must complete prior to release of a given job (PRED). A message is printed at the end of this report which indicates the degree of multiprogramming that can be expected.

-Resource Map (Appendix F). A listing of resources remaining for each time increment is provided by this report, along with a theoretical multiprogramming map. Column headings refer to resources remaining available after assignment of those required to support the scheduled jobs.

TERMINAL CONSIDERATIONS

There are a number of commercial scheduling systems available in the market-place that will perform in a manner similar to APS; however, APS is believed to be the only scheduling system that operates from a terminal. The primary advantage in having capability to operate the scheduler from a terminal is the ability to dynamically reschedule on an as-required basis without interruption of the host computer.

Before discussing the scheduling/rescheduling aspect of APS, it would be beneficial to discuss general advantages and disadvantages of automatic scheduling. First, it is perhaps unfortunate that the term "scheduler" is used to describe the subject system. A more accurate term would be "pre-scheduler" because, basically, that is precisely what happens. By utilizing automated scheduling (pre-scheduling) we can determine the best job start sequence to most effectively utilize equipment resources. Automated scheduling will also allow determination of the best time to start "one-time job" after a normal production day has been scheduled. This is readily accomplished by examining the resource map to choose the time frame wherein resources are available for "one-time jobs." If described in the APS data base, it can automatically be fitted into the current schedule by simply entering its name through the terminal, thereby providing the dynamic rescheduling capability.

A secondary benefit of automatically pre-scheduling a periodic computer workload is the forced attention to detail necessary to properly utilize APS. All too frequently a given work period is scheduled in a most cursory manner. By identifying the requirement to the scheduler, more assurance is given that the proper homework is done before a job is entered into the computer for execution. For example, one of the uses made of the APS system is to simulate a given scheduling period to determine whether more work has been planned than can actually be performed.

Of course, as with all planning, the real world frequently will not allow exact execution. Unexpected job failures, equipment failures or changes in processing priority tend to disrupt the best of schedules. Operational environments that are basically unstable may find that pre-scheduling has limited value in defining optimum job sequence. This is because of the bias thrown in by unpredictable run times. Although this is a disadvantage, it can be compensated for in part by the workload simulation aspect and in part by the ease in which rescheduling can be accomplished. APS retains the last schedule produced so that it is possible at any time to enter, through the terminal, the specific conditions at a point in time and obtain a new schedule in a few minutes (3-5).

CURRENT APS USAGE

The USAMC Automated Logistics Management Systems Agency presently utilizes the APS system in two ways. The first way was addressed briefly above; i.e., used as a pre-scheduling planning tool to insure preproduction considerations are honored. A second and growing use of the scheduler is in the area of "simulation" (Appendix G). In this latter role the scheduler plays an increasingly important role in the area of equipment configuration predictions. These predictions are made in the following manner:

-The workload requirements of a given program are described in terms of transaction type and volume, disk storage requirements and master file activity estimates. These basic statistics are utilized to produce program run time estimates.

-The run time estimates are entered into an APS data base along with other job profile parameters described earlier. Model schedules are then produced which attempt to show a theoretical sequence of work that can occur on a given equipment configuration.

-The reports produced by the scheduler are analyzed to determine whether additional hardware is required or too much hardware is provided for an estimated workload at a given location. For example, if the estimated workload cannot be accommodated on a given configuration, the scheduler reports will show all jobs that can be run and will further list the jobs that cannot be scheduled under the constraints provided. A quick analysis of the output reports will reveal that, perhaps the reasons scheduling cannot occur are a limitation or inavailability of some system resource (i.e., core, tape drive, disk drives, etc.). The overall effect of the "simulation" is a quick look at the impact of additional workload on a given equipment configuration. This methodology has proven to be extremely accurate and effective to date with the qualification that, as with a "normal simulator", the results are only as accurate as the input provided.

APS DATA BASE CONSIDERATIONS

As may be noted in Appendix G, a powerful feature of the automated scheduler is the capability to update the scheduling data base with actual history data. Update is accomplished by using data generated by the System Management Facilities (SMF) feature of the IBM OS. This feature automatically records operational parameters for each job and job step executed. It provides all of the basic information included in the APS data base except the predecessor considerations. Not only is the capability to update the data base to reflect actual history on a recurring basis important, but in addition, those installations that must build the job profile from scratch will find the SMF data a valuable source of information. Since SMF is the source of the machine utilization accounting and billing reports, a secondary benefit is obtained. The APS data base can be used to serve not only the scheduler, but also as the repository of utilization statistics.

APS AVAILABILITY

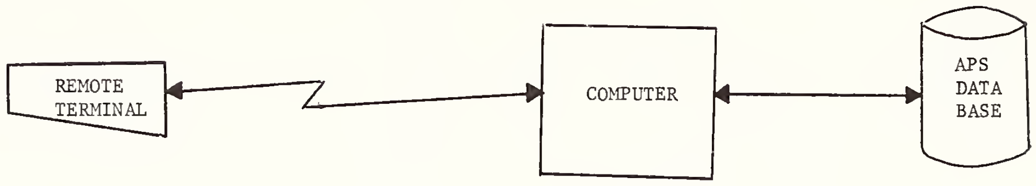
The APS program described in this paper is a proprietary package of Value Computing, Inc. of Cherry Hill, New Jersey. In October 1972, the USAMC ALMSA acquired a license for this package. Provisions in the license include the use of the package throughout the Department of the Army without additional charge.

The program as it stands under the licensing arrangement was designed around the requirements of the CCSS. In this respect, it is built to operate under the IBM MVT operating system Release 20.6 and higher. The access method specified is TCAM (telecommunications access method).

Appendix A
FILE MAINTENANCE

RUN	CORE	RUN	EARLY	LATE	---DEVICE-INFO---			%	TRANS	TRANS	FIELD						
NAME	(1K)	TIME	START	FINISH	CL/DV	CL/DV	CL/DV	UTL	PLUS	MINUS	DESCRIPTION						
DA18	54	5			04/01			10	75		JOB DESC						
											PRED 1-6	DA11	DA12	DA13	DA16	DA17	
											PRED 7-12						
											FREQUENCIES	SD2	SC				
											MASTER FILES						
DA21	54	5			04/01			10	40		JOB DESC						
											PRED 1-6	DA11	DA12	DA13	DA16	DA17	DA18
											PRED 7-12						
											FREQUENCIES	SD2	SC				
											MASTER FILES						
DA22	80	5			04/02			10	76		JOB DESC						
											PRED 1-6	DA11	DA12	DA13	DA16	DA17	DA18
											PRED 7-12	DA21					
											FREQUENCIES	SD2	SC				
											MASTER FILES						
DA23	54	10			04/05			10			JOB DESC						
											PRED 1-6	DA11	DA12	DA13	DA16	DA17	DA18
											PRED 7-12	DA21	DA22				
											FREQUENCIES	SD2	SC				
											MASTER FILES						
DA24	320	5			04/04			10			JOB DESC						
											PRED 1-6	DA11	DA12	DA13	DA16	DA17	DA18
											PRED 7-12	DA21	DA22	DA23			
											FREQUENCIES	SD2	SC				
											MASTER FILES						

Appendix B
FLOW



Appendix C
STATUS REPORT

	Devices			Early	Late	Run	Core	CPU	ID	Job-Strm			
Run Name	CL/DV	CL/DV	CL/DV	Start	Start	Time	*1K	PCT	NO	1ST Run	Pred-1	Pred-2	Pred-3
D06BX	04/01			00.05	23.40	15	250	10	001		D27CX		D20DX
D15DX	04/01			00.05	22.50	65	140	10	004		D15AX		
D15AX				00.05	22.30	20	120	10	005		D38AX	D17AX	Z01ELO
D04AX	04/01	05/01	06/01	00.05	14.45	260	126	10	006		D02AX		
D04BX	04/02	05/01	06/01	00.05	19.05	225	228	10	007		D04AX	D03AX	
D04CX	04/02			00.05	22.50	65	130	10	008		D04BX		
D04DX	04/01			00.05	23.45	10	130	10	009		D04BX		
D04XX	04/02			00.05	23.45	5	490	10	010		NONE		

Appendix D
SCHEDULER

```
0719-01

LIST OF UNSCHEDULED RUNS

*D04XX
```

Appendix E
SCHEDULE

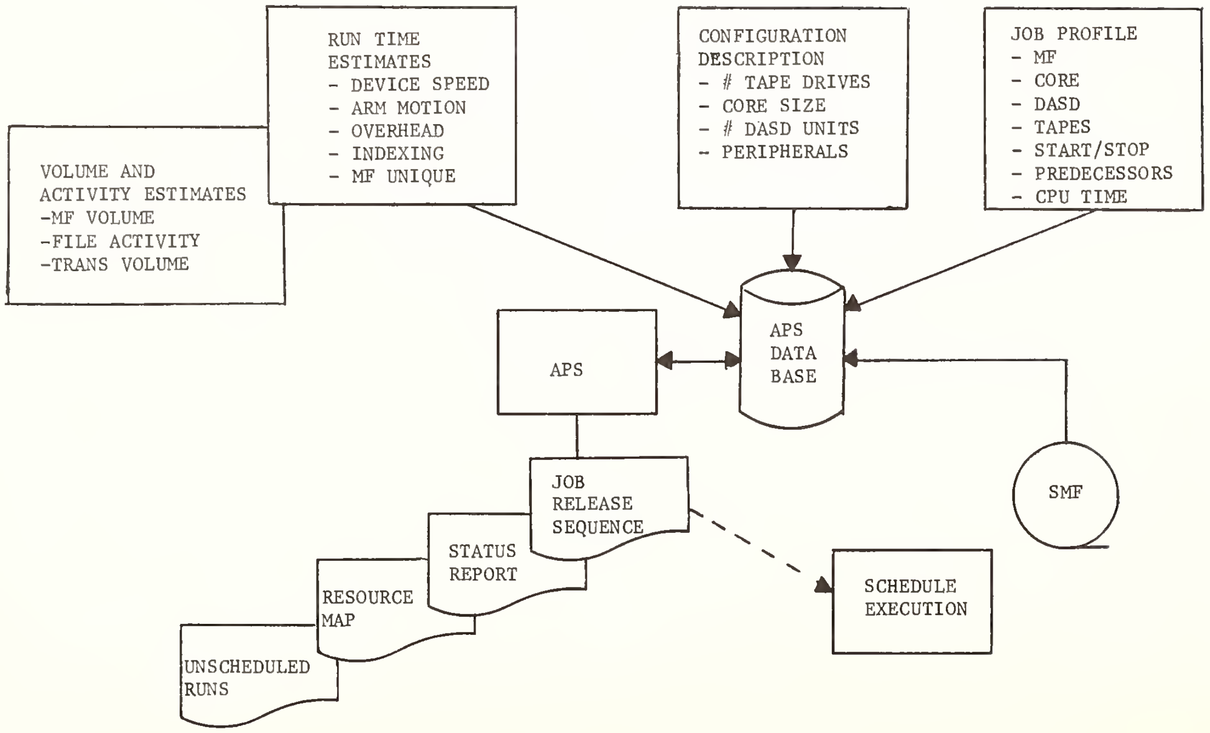
```
Set-Up   Run-Name   Lev   Remove   Time   Str   Pred-1   Pred-2   Pred-3

00.05    D04AX      1     04.25    260          D02AX

00.05    D15AX      2     00.25     20          D38AX    D17AX    Z01ELO

00.25    D06BX      2     00.40     15          D27CX             D20DX

00.40    D15DX      2     01.45     65          D15AX

04.25    D04BX      1     08.10    225          D04AX    D03AX

08.10    D04CX      1     09.15     65          D04BX

08.10    D04DX      2     08.20     10          D04BX


***SCHEDULE COMPLETED***


ELAPSED TIME OF THIS SCHEDULE 09 HRS 10 MIN
TOTAL TIME TO SERIAL PROCESS 11 HRS 00 MIN
```

Appendix F

M/P MAP

TIME	LEVEL1	LEVEL2	LEVEL3	LEVEL4	LEVEL5	LEVEL6	TRANS	COR	REA	PUN	PRI	TAP	FSN	REF	FFC	FIS	FAR	FGL	AHF	RAH	SAD	TRA
00.05	D04AX	D15AX					15974	254	2	1	2	9			1	1	1	1	1	1	1	15
00.10	I	I					15974	254	2	1	2	9			1	1	1	1	1	1	1	15
00.15	I	I					15974	254	2	1	2	9			1	1	1	1	1	1	1	15
00.20	I	I					15974	254	2	1	2	9			1	1	1	1	1	1	1	15
00.25	I	D06BX					15662	124	2	1	2	8			1	1	1	1	1	1	1	15
00.30	I	I					15662	124	2	1	2	8			1	1	1	1	1	1	1	15
00.35	I	I					15662	124	2	1	2	8			1	1	1	1	1	1	1	15
00.40	I	D15DX					15872	234	2	1	2	8			1	1	1	1	1	1	1	15
00.45	I	I					15872	234	2	1	2	8			1	1	1	1	1	1	1	15
00.50	I	I					15872	234	2	1	2	8			1	1	1	1	1	1	1	15
00.55	I	I					15872	234	2	1	2	8			1	1	1	1	1	1	1	15
01.00	I	I					15872	234	2	1	2	8			1	1	1	1	1	1	1	15
01.05	I	I					15872	234	2	1	2	8			1	1	1	1	1	1	1	15
01.10	I	I					15872	234	2	1	2	8			1	1	1	1	1	1	1	15

Appendix G
CONFIGURATION DEVELOPMENT SYSTEM



PERFORMANCE EVALUATION OF TIME SHARING SYSTEMS

T. W. Potter

Bell Telephone Labs

Introduction

One of the most severe problems we face today in a new in-house time sharing system is that it becomes overloaded and degraded overnight. If it were possible to determine under what load conditions time-sharing performance degrades beyond acceptance then one could estimate when to upgrade. More important than knowing when to upgrade is knowing what to upgrade in order to obtain the most cost-effective performance possible. This paper presents techniques that have helped to solve these problems. It begins by describing methods of analyzing how the user is using the time-sharing system. Through these methods one can create reasonable synthetic jobs that represent the user load. These synthetic jobs can be used in benchmarks under elaborate experimental design to obtain response curves. The response curves will aid in determining under what load conditions the time-sharing system will degrade beyond acceptance and the most cost-effective method of upgrading. This information together with a reasonable growth study, which is also discussed, will keep the time sharing system from severe performance degradation.

The response data obtained from benchmarks can be used to create an analytical model of the time-sharing system as well as validating other models. Such models and their usefulness will be discussed.

Some new in-house computer systems can provide batch, remote batch, transaction processing and time-sharing services simultaneously. These sophisticated systems do not lend themselves properly to performance analysis. Therefore, the task of keeping the time-sharing system tuned to maximum performance or minimum response time is difficult. In order to standardize more stringently the techniques available to keep time-sharing response minimal we must separate time-sharing from the other environments or dimensions and present the resulting techniques in this paper. These techniques can then be extended to the multi-dimensional case.

Techniques

Let us assume we are given the task of creating a five-year growth plan for an existing in-house time-sharing system. If we know what kind of users are on the system and how they are using it we would know the usage characteristic or user profiles of the time-sharing system. Now given the estimated growth of time-sharing we could determine the systems reaction to these users.

Therefore if we know the growth (Figure 1) and we know how the time-sharing users are using the system at this time (user profile) then by benchmarking we can determine the system reaction at higher loads and obtain a response curve (Figure 2). This assumes of course we can define response time. If we re-benchmark using different configurations we can then see the effect on performance of alternate configurations.

Figure 1

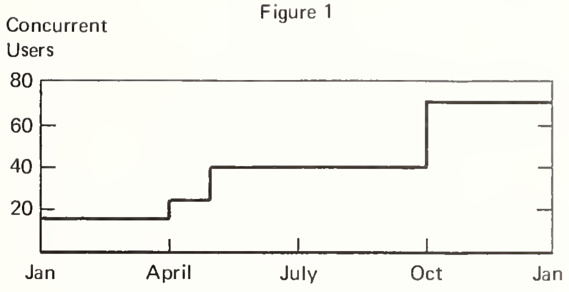
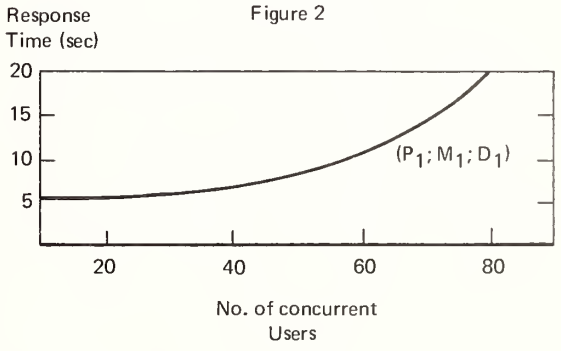
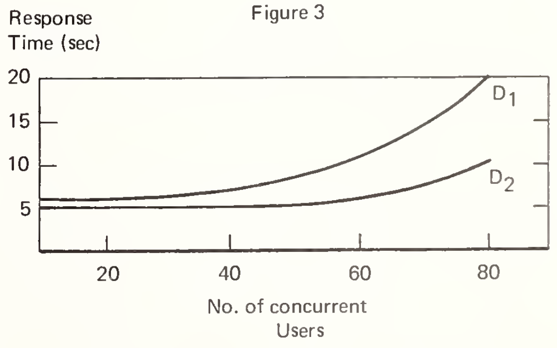


Figure 2



Suppose that the curve in Figure 2 is based upon a system with processor P_1, memory M_1 and disk D_1. Then we re-benchmark using the same processor P_1 and memory M_1 but change disk D_1 to D_2. Now we obtain curves like Figure 3.

Figure 3

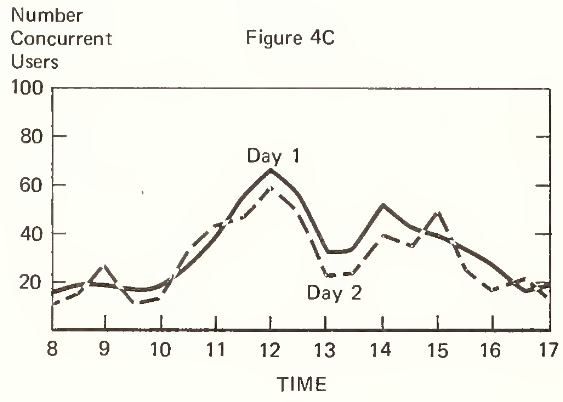


We could re-benchmark again changing any number of possible hardware or software factors. Hardware factors are discussed in this paper.

The hardware monitor can collect information relating to the instruction mix which can be very helpful in reproducing the processor requirement. The monitor can also collect information relative to the terminal user in terms of average number of characters received/transmitted per user interaction. Since the monitor can also collect information relative to the peak periods of utilization we can determine when to study the log files. In order to guarantee performance one needs to reproduce the peak period loads and therefore information on non-peak periods need not be used in most cases.

Typically the log files give information relative to average memory requirement, average processor time and average disk and terminal I/O. Most log files account for the time-sharing processor time, disk and terminal I/O and number of commands or languages requested. For example, if Fortran was used then those resources used by Fortran during the sample period would be recorded. Typically by studying the commands or languages we can find a small number of them that account for the majority of the resource utilization. A user survey is very helpful in determining the most frequent programming languages, the program size in arrays versus code, and the types and sizes of disk files. The console log usually gives an indication of the number of concurrent users during different periods for an entire day. Console log information can be used to make sure that the time-sharing system is relatively consistent from day to day. This can be done by graphing the maximum number of concurrent user during each hour of a day and comparing the daily graphs. (See Figure 4C). If large fluctuations do occur then one must compare the user profiles that are created during the different peak periods and determine which profile to use as a pessimistic representation of the user load (a harder load on the system).

Figure 4C



These four major tools (monitors, log file, surveys and console logs) now allow us to create one or more synthetic jobs that reasonably represent the user load. A synthetic job is like any other job in that it requires a programming language, processor time, disk and terminal I/O, and memory requirements. If more than one programming language has large usage then more than one synthetic job is required. Let us work through an example on how these tools are used to create synthetic jobs. Data contained within this example are purely hypothetical.

Let us assume that a hardware monitor has been connected to the processor(s), channel(s), and communication controller or processor. This monitor should at least collect utilization data to determine peak periods of resource utilization. The monitor should also collect or sample the instructions (op codes) executing within the processor(s). The monitor could also collect terminal related data by sampling a register within the communication processor which contains information on each terminal I/O. This information could consist of the terminal number; whether the terminal was presently sending or receiving data; and the number of characters transferred. The terminal data could also be collected via a software monitor. Now let us assume that there is only one synthetic job which will represent the entire load. What might we observe as typical hardware monitor data?

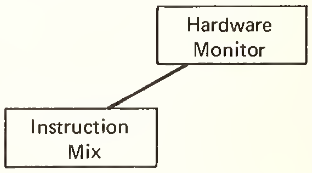
In creating the synthetic job we need to determine not only the amount of processor time used but also the type of instructions executed (scientific or data processing). We might observe the following hardware monitor data for instruction mix.

OP Code Usage in Percentage

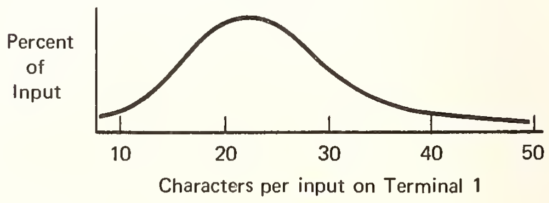
OP Code	Day 1	Day 2	Day 3
Load	29.6	28.7	30.7
Store	19.7	19.9	19.4
Compare	13.5	10.0	12.1
Transfer	25.5	24.0	24.9
Arith	7.8	7.9	7.1
MISC.	3.9	9.5	5.9

This data might indicate to the analyst that the users are data processing oriented as opposed to scientific.

Schematically, we have used the monitor to collect data on the instruction mix:



Now by obtaining terminal data we could collect the following distributions on each terminal:



Let us assume that disk D_2 produces the desired minimum response time. Response time is defined as the time it takes the system to respond with the first printable character after the user returns the carriage on his terminal. We shall assume that the system was originally engineered for minimum response time of say 2.5 seconds. This assumption creates a number of unanswered questions. What expenses are associated with the best response time as compared to 'good' response time (2.5 seconds vs. 5 seconds)? At what response time does the user become irritated? The answers to these questions might allow the system to be engineered in a more cost-effective manner.

Let us assume that an average response of ten (10) seconds or greater irritates the users. Our goal then in this five-year growth plan is to keep average response time below ten seconds. Note that human factor methods must be utilized in order to obtain the unknown irritation level.

This irritation level would indicate in Figure 4A that at 45 terminals the configuration needs upgraded possibly from disk D_1 to disk D_2. This decision to upgrade is based only on performance. One could base a decision to upgrade on cost-performance, in other words it might be cheaper to accept a penalty in dollars for responses beyond 10 seconds up to say 15 seconds before replacing the D_1 with a costly disk D_2. This assumes of course one can attach a price to degradation.

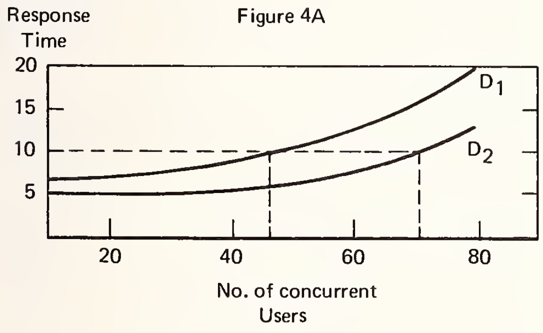


Figure 4A

Now given the magic number of 45 (the point to upgrade), Figure 1 indicates that the disk D_2 should be on the system in October.

Let us assume that the in-house time-sharing system will perform adequately with the new disk D_2 until the growth exceeds 72 concurrent users (Figure 4A). In order to keep response time below ten seconds when the number of users exceed 72 we must reconfigure the hardware and re-benchmark. We might find that an additional module of memory M_2 provides the needed response time as in Figure 4B.

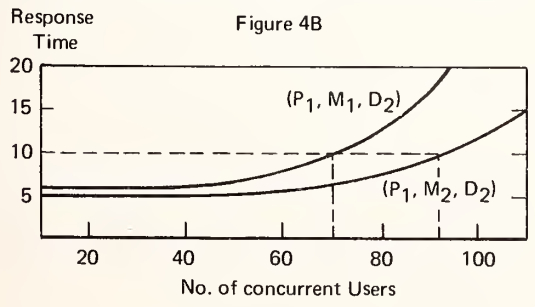


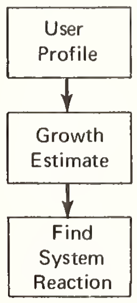
Figure 4B

From Figure 1 we see that the additional module of memory should be on the system by January.

This process of benchmarking and considering alternative configurations can be continued over and over until a reasonable five-year plan takes shape.

Therefore given the usage characteristics of the present in-house time-sharing users and the growth estimates we can find the systems reaction and produce a rgowth plan.

This can be represented schematically as:

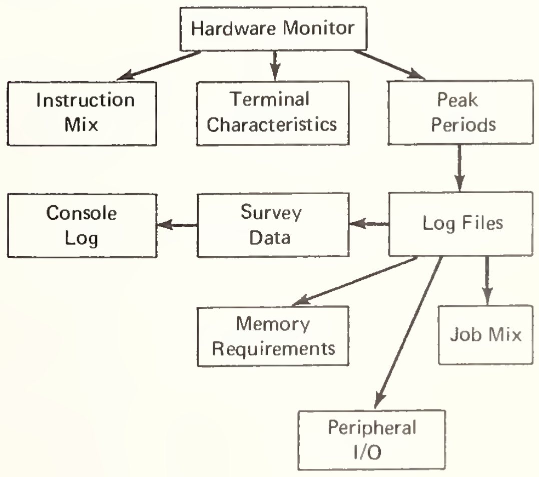


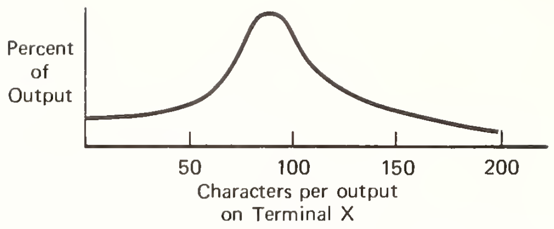
The major limitation to such an ideal scheme as outlined above is that we usually are not given the usage characteristics or the growth estimates.

Therefore, we must determine the user profile and growth estimate and then proceed finding the systems reaction.

User Profile

The user profile or characteristic can be obtained via hardware monitors, log files, survey data and console logs. Note that software monitors can also be used but are not addressed in this paper. These tools interrelate in the following manner:





These distributions could be summarized as follows:

Term No.	Avg. Input	Avg. Output	Percent In	Percent Out
1	18	100	3.58	4.97
2	29	76	2.93	6.11
.				
.				
.				

Total Input - 30%

Total Output - 70%

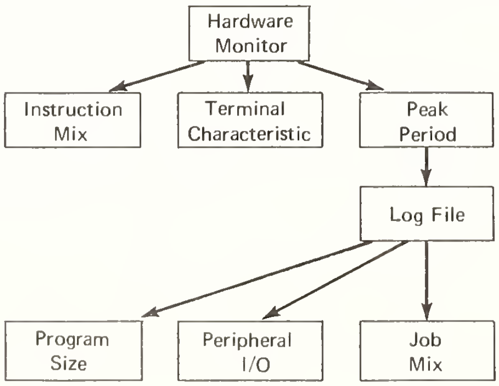
Weighted Avg Input - 30 characters

Weighted Avg Output - 90 characters

Collecting this data over three or more days would allow the analyst to conclude that the terminal user is data processing or report writing at his terminal.

In order to obtain detailed data for the synthetic program we need to determine the peak periods of resource utilization. Here we might find that the peak period is from 2:00 p.m. to 3:00 p.m. each day. We could average the synthetic job characteristics over the sample days or create the job characteristics from the day where most resources were used. Assuming the latter case, we could then extract from the log files the peak period data.

We need to determine how large the synthetic program should be and how much peripheral I/O is generated. Schematically the log file and hardware monitor relate as follows:



On some log files the average program size for each language processor can be obtained while on other log files only the overall average program size exists for time-sharing jobs. If Fortran and Basic are the most frequently used languages then an average size Fortran and Basic job needs to be created as synthetic jobs. Let us assume the log files indicate that the average Fortran programs object code is 12K words (K=1024) on some machine and the average BASIC program's object code is 4K words. We might find that Fortran is executed n times during the peak period and Basic is executed m times according to the log file. With this information and the number of disk I/O's performed per execution of Fortran or Basic we might find that Fortran has 30 disk I/O and Basic has 50 disk I/O's on the average. If possible we should separate disk I/O's into input and output.

In time-sharing we often find that program execution only occurs a small percentage of the time (30%) whereas command execution occurs most frequently (70%). Therefore, we must study all the possible commands (old, new, list, etc.) and select a subset which will account for a large percentage of resource utilization. In many time-sharing systems where the users are data processing oriented the three commands: request a file; edit a file; and list a file, account for the majority of command related resources. This, however, forces the analyst to create three synthetic jobs for the commands and one for each language used. Therefore, if Fortran and Basic are highly used then there are two more synthetic jobs or five in all.

Typically, the time-share commands process file and the files can be reproduced. For example listing a file by the LIST command can be reproduced by obtaining an average file size (to recreate the terminal and disk I/O) and the average processing time required to list the file. Each command is studied in order to reproduce it as closely as possible. Let us assume that five synthetic jobs are created which are data processing oriented then the next question is how are these five synthetic jobs mixed together? In order to determine how the synthetic jobs are mixed together we use the log files to determine how often each command or language is executed. We sometimes see that these five jobs account for 90% of all system resources but are only referenced 40% of the time. If we assume that these five are the only jobs on the system and adjust the 40% figure to 100% we obtain the frequency of executing the various jobs. For example, we might find:

Basic	- 14%
Fortran	- 10%
List	- 20%
Edit	- 10%
Old	- 46%

This percentage can represent the steady state probability of being in any one synthetic job and as such represents this job mix. This assumes we can define steady state. From a hardware monitor and the log files we can reasonably create synthetic jobs and determine their mix. There are still a few unanswered questions like what kind of files do the programs use (random or sequential)? This question and others can be reasonably answered via the user survey. The survey helps to fill gaps with synthetic jobs and should be designed to find out what the user thinks his program does.

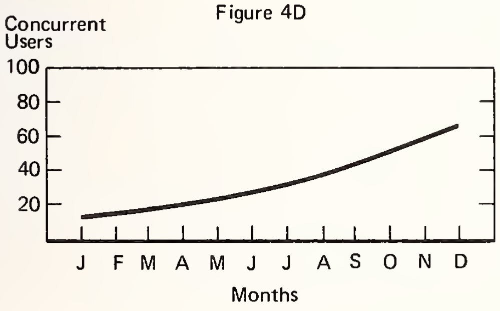
The console log allows the analyst to determine how consistent the users are from day to day and how often the system goes down. The later item is important because a false peak period is usually observed when a system regains consciousness and as such should be ignored.

After creating the synthetic jobs by studying the usage characteristics we now must determine the anticipated growth.

Growth Study

An in-house time-sharing system grows because: new applications are created on the system; more users become familiar with the in-house system; and outside users (commercial service bureaus) are requested by company policy to move in-house whenever possible. In the first two cases the anticipated growth of time-sharing is directly related to the amount of money in the budget. Actually we assume that increased budget money means increased usage but we do not actually know whether a $2000/month increase equals 10 concurrent users or 2 concurrent users. For this reason we must collect past budget estimates and relate them to past usage. For example we might find a 10% increase in dollar usage in the past year (hopefully budgeted) and a real time-share growth like Figure 4D which might indicate a 20% growth in concurrent users.

Figure 4D



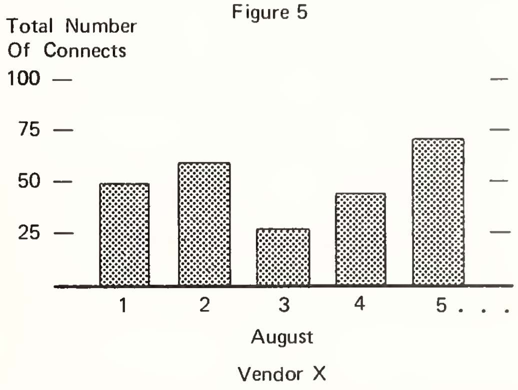
Now our anticipated growth without service bureau users moving inside is estimated at 20%. What happens to this anticipated growth if outside users are requested or desire to move inside?

If this in-house system is regulated in such a manner that the outside users can only move in-house by going through designated channels then a regulatory body can control the percentage of outside users to move inside. This percentage would be based upon the number of concurrent users from your company on each of the service bureaus. Once this percentage has been obtained then the regulatory body can limit the number of outside users to move inside or plan the move over a period of time. The techniques used to obtain the above percentage can be used to solve an entirely different problem. This is the problem of predicting an initial cutover load on a new time-sharing system when all users are outside on commercial service bureaus. Since we assume the in-house system already exists we will present the methods required to determine the above regulatory percentage.

First, obtain past bills for time-sharing services charged to your company. Some of these bills will contain information on log-on/log-off time by user-id. Other bills will only have dollar resources used and each billing unit will be different.

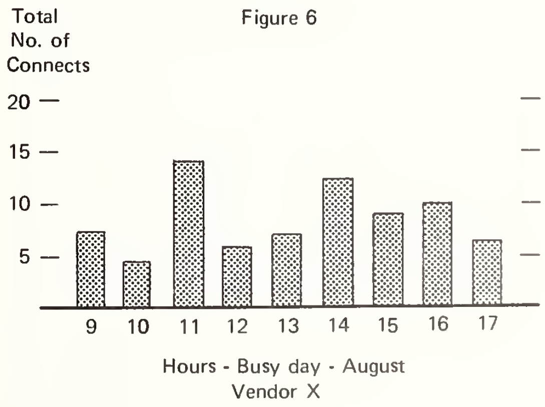
If we were to graph the number of connects on each (if possible) of the time-sharing service bureaus by days within each month (Figure 5) we can obtain peak days within each month. A connect is where a user logs onto the time-sharing system.

Figure 5

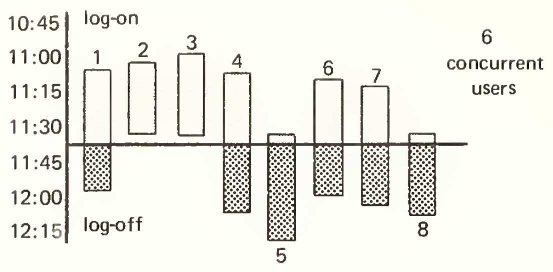


Now by observing, which day in the month was the peak day we can graph the total number of connects by hours on that day (Figure 6).

Figure 6

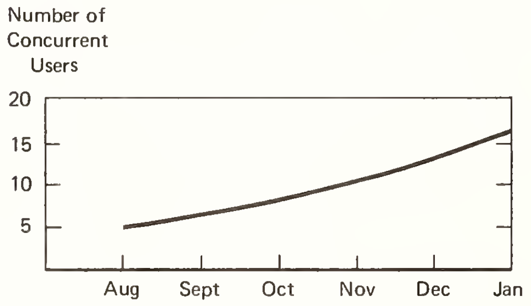


The hour given on Figure 6 with the largest number of connects should be studied in more detail. If we are given the log-on time and elapsed time we can compute the log-off time and create the following time diagram:



This now says that for the month of August we could see as many as six concurrent users from your company on this system at one time.

We can obtain reasonable estimate of current outside growth like Figure 7 for each vendor by studying their billing data over a four to six month period.

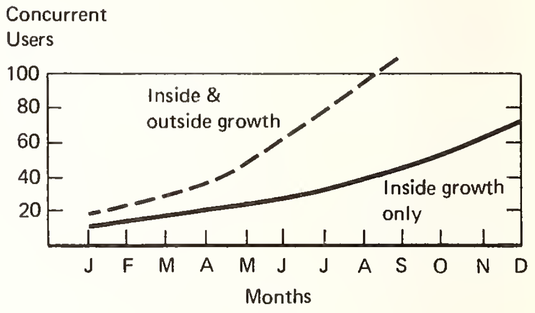


This might indicate that your company has an estimated outside growth of 25% on vendor X.

We could now tabulate the outside growth in the following manner:

Service Bureau	Present Concurrent Users	Predicted Growth
X_1	5	25%
X_2	10	10%
:	:	:
X_n	7	15%

If the regulatory group (limits outside to inside moves) has two people to help move the outside users inside then they must be scheduled according to some predetermined plan. This plan might indicate that two concurrent users from vendor X_1, one from vendor X_2 and so on can be moved inside per month according to available manpower. If all time zones are the same we can assume that our in-house peak number of concurrent users will increase by the two users from vendor X_1, one from vendor X_2, etc. which move inside per month. The number of concurrent users moving inside now allows us to determine the regulatory percentage discussed earlier as say 20%. We now conclude our growth estimate with the following growth chart:



This chart indicates that if the outside growth moves inside and the inside system observes some regular increase then the system soon gets out of hand. This could mean that the schedule of moving outside users inside namely the regulatory percentage of 20 is too progressive.

One of the main reasons for discussing outside time-sharing usage is that a company which has inside time-sharing probably has some outside usage. This outside usage usually grows as does the inside usage since more computer users seem to be turning to time-sharing. For this reason any growth plan for in-house time-sharing would be incomplete without some consideration of outside users migrating inside.

The growth procedure discussed above will give us a reasonable estimate of growth in the immediate future. However what happens in the long term is never known. Setting up this procedure will allow you to follow your growth and at least estimate out at three to six month intervals.

System Reaction

We can find the systems reaction by setting up experiments on a controlled machine and collecting performance data like response time. Experiments to collect response time data can be set up using a terminal emulator which makes another computer look like 'n' terminals. This of course assumes that we can recreate the load via synthetic jobs in a mix (20% job 1, 10% job 2, etc.). For example we can start 20 terminals executing jobs for say one hour during which we can force synthetic job one to execute 20% of the time.

The terminal emulator is designed to time stamp and store all data sent or received onto a magnetic tape for later analysis. During the one hour period to run 20 termianls we must allow the system to stabilize. This stabilization can be defined in a number of ways, however I find it convenient in data analysis to run the system long enough for the jobs to reach a minimum variance on their individual response times. For example synthetic job one has a mean response time and a variance. Once the system is stable the variance on response time for synthetic job one is minimal. The time required to reach minimal variance for synthetic job one will differ from the other synthetic jobs assuming of course steady state exists. Note of course the difficulty in observing a running variance and deciding on steady state.

It is extremely hard to analyze the effect of five synthetic jobs and 20 terminals with a measurement criteria like response time unless one analyzes each synthetic jobs response time as a function of the mix. The raw data should always accompany the analysis in the following manner:

Synthetic Job No. 1

	Observation				
Terminal	1	2	3	4......	N
1	X_{11}				
2					
3			Measurement Criteria		
⋮					
20					

This table of raw data indicates that we observed the response time or measurement criteria for synthetic job one on each terminal a number of times. Keep in mind that at the same time synthetic job one is running it is possible for the other jobs to be running.

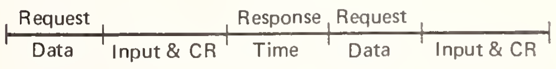
Let us assume we have determined a growth estimate and a user profile. By benchmarking alternate configurations with varying number of concurrent users up to the maximum predicted by the growth estimate we can obtain various response curves. These curves will then be useful in determining when to upgrade. Since the growth estimate may only be reliable from three to six months in advance then these curves should be very accurate for that period. Now if we can be assured that as the growth increases the user profile stays reasonably the same then the curves can be useful throughout the entire growth period.

One should not benchmark every time the user profile or growth estimate changes. Typically a by-product of benchmarking will be an awareness of how sensitive the response curves are to certain factors (e.g., processor requirement). Those factors that make up the synthetic jobs should be tracked or followed on the in-house time-sharing system so as to observe any great change. This usually can be done via a log file analysis program. If the user profile changes greatly and there is not a clear cut alternative for a system upgrade then re-benchmarking is usually required. Therefore by determining a user load and growth estimates we can analyze the systems reaction and as a result upgrade our system based on cost-effectiveness. There are a number of problems not yet addressed in this paper, for instance if there are five synthetic jobs (three commands, two jobs) what is response time? Another interesting problem is how do we determine how much better disk D_2 is than disk D_1 in Figure 3? One way is to eye-ball the curves and estimate, but this does not account for the variance on each point on the two curves. For example, we will not find at 40 terminals on disk D_2 a response of exactly 4.5 seconds. We will find a mean of 4.5 plus or minus some deviation. Since some variance exists for 40 terminals of D_2 some variance also exists for 40 terminals of disk D_1. Could this indicate that there is no difference between disk D_1 and D_2 at 40 terminals? It depends on the variance of the two means typically referred to as statistical significance testing. How do we take into account the variance on D_2 at 40 terminals and at the same time 60 terminals since the variances will also differ? One way is to create an adequate statistical experimental design for the benchmark so that a regression analysis or analysis of variance can be performed on the data. The advantage of regression analysis is that it can relate the difference between the two curves in terms of additional terminals at the same response. For example, disk D_2 in Figure 3 may give 40 more terminals at the same response time than the disk D_1. The 40 more terminal figure is plus or minus say 2 terminals. This is very useful information because if disk D_2 cost $100 more than disk D_1 per month then it cost $2.50/terminal to replace disk D_1 at 40 terminals with D_2. Comparing the $2.50/terminal for the disk to say $5.00/terminal for a memory increase we would find the disk enhancement to be more cost-effective. This regression analysis can be very useful in comparing curves but it completely depends on the benchmark and how well the experiment was designed. I recommend reviewing your design with statisticians prior to running the benchmark. Therefore by creating a user load, a growth estimate and an adequate experimental design we can make a reasonable growth plan for a time-sharing system. This procedure for a growth plan is repetitive and must be so in order to be reasonably approximate.

The techniques presented in this paper to recreate the load, estimate growth and find the system's reaction were not designed to be totally encompassing primarily because the methods of data analysis and decision making in performance evaluation are barbaric.

The remainder of this paper addresses the real life problems associated with using response time as a measurement of time-sharing performance. In this we shall assume that the load can be reproduced via synthetic jobs and accurate growth forecasts can be made. With these assumptions we can design experiments in order to collect response curves like Figure 3. A terminal emulator or load driver would be used to implement these experiments. The real problems are in the benchmark results. Let us study the curves in Figure 3.

First give response time a simple definition. Create a synthetic job written in Fortran with a 'DO' loop in which a terminal read statement is placed. In program execution data is requested from the user; the user inputs his data and hits a carriage return, then the program requests more data. This may be viewed on a time scale as:



Therefore response time is the time from a carriage return 'CR' to the time the system responds with a printable character. We know as a fact that if the 'DO' loop requests ten entries of data from the terminal then we will have ten different response times. Now if this one terminal executes the synthetic program over and over we will obtain yet different response times. Ten different terminals all executing separate copies of this synthetic program will also observe different response times.

Three separate sources of variability exist: within the synthetic job; across executions of synthetic jobs within terminals; and across terminals. Yet this is the simplest definition of response time possible. Now Figure 3 looks like Figure 7 where with each mean response time plotted we have also plotted a window or variance around the mean. Therefore Figure 7 looks like Figure 8.

Figure 7

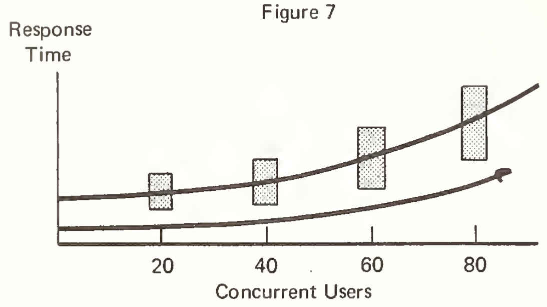
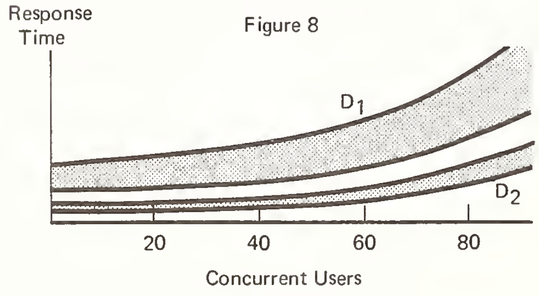


Figure 8



The problem is that we usually show our management how different curve D_1 and D_2 are from Figure 3, when in fact the real difference between D_1 and D_2 is shown in figure 8. If we were to base a decision to upgrade on Figure 3 we could really make a mistake. Instead we should consider mean response time and variance. Curve D_1 in Figure 8 is now a very fat curve or interval. This interval comes about because we want to be sure that the mean at Y terminals is in fact within the interval a certain percent of the time. For example if we want to guarantee 95% of the time that the mean response time at 40 terminals is a certain amount we could create an interval in which this mean would lie 95% of the time. The more you want to guarantee a mean response time the wider the interval. Therefore in order to guarantee a difference between the curves in Figure 8 we must use both the mean and variance at each possible point on the X-axis. We are now faced with comparing intervals. Now the problem of determining when to upgrade and what to upgrade is more complex. We could compare alternative curves via regression analysis and if given the cost of various alternatives we could create econometric models to determine the most cost effective upgrade. In addition, if a 95% guarantee is not needed then one might be willing to risk an 80% or 51% guarantee in which case risk analysis or statistical decision theory becomes useful. Keep in mind we have defined response time in the simplest way possible but it really doesn't represent the real world because a single synthetic job usually cannot recreate the load. What then . . . multiple synthetic jobs?

If we have more than one synthetic job or multiple synthetic jobs how is response time defined? If we are to recreate the real load on the system then these synthetic jobs must have different response times. For example one program may accept data, compute and then respond where as another may accept a command and immediately respond. This source of variability between different jobs is usually so large that response time has no real meaning. One could create a graph like Figure 8 for each of the synthetic jobs. Now the response at twenty terminals would mean that when twenty terminals were connected to the system running the mix of synthetic jobs we obtained a mean response time of x for synthetic job number one with a variance of Y. We are now faced with the problem of determining with not one synthetic job but many synthetic jobs when to upgrade and what to upgrade. One might find for synthetic job one that a new disk is required after 40 concurrent users whereas synthetic job two indicates more memory at 35 concurrent users. Well we could get around this problem by some cost trade-offs but we immediately land into another pit. We assumed that the multiple synthetic job recreated the load, today. What about tomorrow? Will our decisions be good if synthetic job one changes in its compute requirements? How good?

All these problems because we used response time as our measurement criteria in time-sharing. We can't use processor utilization as the measurement criteria so what now? Most of us will ignore variances and return to the simple definition of response time with one synthetic job and results like Figure 3. If this is the case then the technique outlined for a time-sharing growth plan are quite adequate. If we wish to use variances and multiple synthetic jobs then we must first decide how to use the data. Then we can proceed to make the growth plans more rigorous.

A CASE STUDY IN MONITORING THE CDC 6700 - A MULTI-PROGRAMMING, MULTI-PROCESSING, MULTI-MODE SYSTEM

Dennis M. Conti, Ph.D.

Naval Weapons Laboratory

ABSTRACT

The complexity of many present day computing systems has posed a special challenge to existing performance measurement techniques. With hardware architectures allowing for several independent processors, and with operating systems designed to support several classes of service in a multi-programming environment, the problem of measuring the performance of such systems becomes increasingly difficult. The architecture of the CDC 6700 poses such a challenge.

With two CPU's and twenty peripheral processors (independent, simultaneously-executing processors), monitoring the CDC 6700 becomes an exceptionally difficult task. With the operating system supporting four modes of service - batch (local and remote), graphics, time-sharing, and real-time - all in a multi-programming environment, the monitoring task becomes even more complex.

This paper presents a case study of an on-going effort to monitor the CDC 6700. The goals, approach, and future plans of this monitoring effort are outlined, in addition to the benefits already accrued as a result of this study. Several software monitors used in the study are discussed, together with some proposed hardware monitoring configurations.

The performance measurement study described here has proved to be an extremely worthwhile venture, not only in terms of its direct impact on improved system performance, but also in terms of "spin-off" benefits to other areas (benchmark construction, measurement of operator impact, feedback to on-site analysts and customer engineers).

I. Background

As a large R&D center with requirements for batch, interactive, and real-time computing, the computational needs of the Naval Weapons Laboratory (NWL) are particularly demanding. A CDC 6700 computer with a modified SCOPE 3.3 operating system is currently in use to support these needs. In order to fully appreciate the complexity of monitoring such a system, a brief description of its hardware and software architecture is in order.

The CDC 6700 consists of two CPU's (CPU-A approximately three times faster than CPU-B), and twenty peripheral processors (PP's). The peripheral processors are virtual machines with their own CPU and memory, operating independently of each other and the two CPU's. The PP's may access both central memory and their own 4K of core. Central memory consists of 131,000 60-bit words. Twenty-four 841 disk drives, three 844 disk units, and two 6638 disks account for over 970 million characters of permanent file space and over 360 million characters of temporary scratch space.

The modified SCOPE 3.3 operating system supports four concurrent modes of service - batch (local and remote), graphics, time-sharing, and real-time. The time-sharing subsystem, INTERCOM, provides both a file-editing and a remote batch capability. The graphics subsystem supports two graphic terminals via a pair of CDC 1700 computers. The real-time subsystem provides a hybrid computing capability in addition to other real-time applications.

Up to fifteen jobs may be active at one time. Each active job is said to reside at a "control point" and may be in one of five stages of execution (executing with one of the CPU's, waiting for a CPU, waiting for some PP activity to complete, waiting for an operator action, or rolled out). Both CPU's may never be assigned to the same control point at the same time (i.e., the operating system does not support parallel-processing of a single job).

The system monitor, MTR, resides in one of the PP's and oversees the total operation of the system (scheduling the CPU's, scheduling the other PP's, honoring CPU and PP requests, advancing the clock). As there are no hardware interrupts in the system, all requests for system resources are done through MTR. Of the remaining nineteen PP's, some have fixed tasks assigned to them (e.g., one is dedicated to driving the operator's display), while the others are available for performing a wide range of tasks (input-output, control-card processing, job initiation). A PP program may reside either on the system device or in central memory. When an available PP is assigned a task by MTR, the relevant PP program is loaded into the PP memory and execution begins. Upon completion, the PP is again available for a system task.

Clearly, the complexity of both the hardware and software architectures of the 6700 poses a tremendous challenge to existing performance measurement techniques. In light of the "multi" aspects of the system (multi-programming, multi-processing, multi-mode), not only is the acquisition of monitoring data difficult, but its interpretation is even more so.

In spite of these seemingly overwhelming difficulties, such a monitoring effort was undertaken at NWL in the fall of 1972. The goals of this effort were to:

1) determine any existing bottlenecks in the system;
2) provide a day-to-day "thermometer" with which any abnormal aberrations could be detected;
3) aid in the planning of equipment configurations;
4) aid in determining the need for and in the selection of new equipment.

What follows is a description of: (1) the sequence of events leading up to the monitoring effort; (2) the monitoring effort itself; and (3) the results and future plans of the effort.

II. The Pre-monitoring Analysis

Prior to the monitoring effort, serious consideration was given to: (1) available monitoring tools; and (2) the system activities to be monitored. Several software monitors were considered. Due to its flexibility, scope, and availability, the software monitor 1SA written at Indiana University was chosen as the primary software monitoring tool. 1SA resides in a dedicated PP and samples various system tables and flags. Due to some basic design differences between the hardware and operating system at Indiana University with that at NWL, approximately three man-months of effort was required to implement 1SA and make its data collection and analysis as automatic as possible.

A second software monitor was written to extract information from the system dayfile (a detailed, running log of major system events). And finally, a third software monitor which records individual user CPU activity was acquired. Termed SPY, this routine resides in a PP for the duration of a user's job, while continually "spying" on his control point.

Choosing the type of system activities to monitor was one of the more difficult parts of the pre-monitoring analysis. Without knowing a priori what the bottlenecks in the system were (if, indeed, any existed at all!), the choice of activities to monitor was based primarily on an intuitive feeling (on the part of several systems personnel) as to the major points of contention in the system, backed up by some rough empirical data. Some of the activities to be monitored were determined in part by the particular monitors chosen. Other activities were found to be best monitored by hardware monitoring.

Before software monitoring could take place, changes were required of 1SA and its associated analysis program to record information on: two CPU's; 131K of central memory; 29 mass storage devices; additional control point activity; and finally, the number of absolute, CPU program (e.g., the FORTRAN compiler) loads. Due to limitations in the size of PP memory, several items recorded in the original 1SA had to be deleted in order to facilitate the above changes. In addition, the output medium of 1SA data was changed from punched cards to mass storage files.

Because several critical items in the system could not be software monitored (e.g., memory conflicts), a feasibility study was undertaken to employ a hardware monitor to capture some of the data. Relevant probe points were acquired, with details of the following hardware monitoring experiments completed by 1 April 1973:

1) CPU state;
2) CPU idle vs. I-O activity;
3) number of seeks, number of transfers, average seek time, average length of transfers for all mass storage devices;
4) CPU and PP wait time due to memory contention.

III. The Monitoring Effort

Full scale use of 1SA began in May, 1973. The following describes the data collection and analysis procedures of 1SA as they were and still are being used.

1SA is automatically loaded into a PP each time the system is brought up - whether the first time each day or after a system malfunction. The system is normally up 24 hours a day, six days a week with two hours a day allocated for software maintenance and two hours for hardware maintenance. Input parameters to 1SA allow for n batches of data to be dumped, where each batch represents m minutes worth of monitoring data. After some experimentation, n and m were set to 24 and 60 respectively - thereby minimizing the number of dumps, but yet still maintaining a reasonable sampling period.

A GRAB program is run once a day to collect the dumped batches of data and consolidate them into one large file. An ANALIZE program then reads this file and produces a listing of meaningful statistics. Some system activities recorded by 1SA as a function of time are:

1) percent of time interactive users were editing text vs. executing a program;
2) average and maximum central memory used by interactive users;
3) central memory utilization;
4) CPU utilization (for both CPU-A and CPU-B);
5) control point activity (e.g., percent of time n control points were waiting for a CPU);
6) percent of free space on each of the mass storage devices;
7) average and maximum number of I-O requests outstanding for each mass storage device;
8) PP activity;
9) number of PP program loads;
10) number of absolute CPU program loads.

An additional option was subsequently added to ANALIZE to summarize all 1SA data collected within a given hourly time frame between any two given dates.

The dayfile analysis program was and still is being run daily. Data collected by the dayfile analyzer includes: turnaround time and load statistics for the various batch classes, frequency of tape read and write errors, abnormal system and user errors, frequency of recoverable and unrecoverable mass storage errors.

The SPY program was made available to the general user population in December, 1972. Although its use was originally limited to a few users, others immediately began to see its worth. Requiring only two control cards to execute and produce a CPU distribution map, SPY became extremely easy for the average programmer to use.

After some initial problems with mating probe tips to probe points were overcome, several system activities (CPU state, memory contention) were successfully hardware monitored and their respective probe points verified. This verification procedure required two weekends of dedicated machine time. As a result of this feasibility study, an effort was immediately undertaken to acquire a dedicated, on-site hardware monitor. At the time of this writing, attempts to acquire such a monitor are still proceeding.

IV. Results

The number and type of benefits accrued as a direct result of the monitoring effort unquestionably proved its worth. Some of these benefits were expected, while others were rather pleasant surprises.

The dayfile analysis program proved to be an especially worthwhile tool for detecting abnormal hardware and software errors. Upon noticing such abnormalities, the personnel monitoring the data would immediately notify the on-site analysts or CE's, who then undertook corrective actions. Each week the daily turnaround time and load statistics would be summarized and compared with data from previous weeks. A relative increase in turnaround time and decrease in number of jobs run per day was usually directly proportional to the number of system interruptions. These numbers were thus good indicators of the relative "health" of the system.

The SPY program did provide some dramatic results. One particular CPU-bound program, for example, was found to be spending over 25% of its execution time in the SIN and COS routines.

The hardware monitoring study demonstrated the feasibility of an expanded hardware monitoring effort. In addition, it emphasized the need for the cooperation of local CE's in helping to develop and verify probe points, and for an extensive pre-monitoring analysis period.

The most fruitful results of the monitoring effort undoubtedly came from the 1SA software monitor. At least seven major benefits were directly attributable to 1SA data collection:

1. After the first two weeks of running 1SA, it became apparent that a number of PP programs were continually being read from disk and loaded into a PP (some on the order of 8 times a second). These most frequently loaded PP programs were immediately made central memory resident, thereby significantly reducing traffic on the channel to the system device, in addition to decreasing the PP wait time.

2. With the daily runs of 1SA indicating the amount of free permanent file space, a means existed for assessing the permanent file space situation. When space appeared to be running out, appropriate measures were taken <u>before</u> space became so critical that all operations ceased (as was sometimes the case in the past).

3. Two situations arose which dramatically showed the merit of 1SA as a feedback mechanism to on-site analysts and CE's. Immediately after the introduction of a new mass storage driver, 1SA data indicated that a certain PP overlay was being loaded on the average of 74 times a second - almost as frequently as the other 400 PP routines combined! The analyst who implemented the driver was easily persuaded to incorporate the overlay into the main body of the driver, thereby eliminating unnecessary loads.

   On another occasion 1SA data indicated that one particular scratch device was being used half as much as its identical counterpart. In discussing this situation with one of the CE's, it was learned that he had installed a "slow speed" valve in the unit while waiting for a "high speed" valve to be shipped. He had neglected mentioning this action to any of the operations' personnel.

4. Several times throughout the monitoring period the effect of operator interaction on the system was clearly reflected in the 1SA data. One specific example which occurred intermittently was an operator error in configuring the scratch devices. Specifically, the operator was told to 'OFF' a particular scratch device. However, 1SA data showed that scratch <u>was</u> being allocated to that device. The operator was notified and corrective action was taken.

5. A few months into the monitoring effort three 844 disk drives were installed. With a storage capacity and transfer rate approximately three times that of an 841, their impact on system performance was anxiously awaited. Summary runs of 1SA data collected before and after the introduction of the 844's showed, in fact, a 10% increase in CPU utilization.

6. With the introduction of the 844's, a method was also needed for determining where to put the system device. Should it reside on an 841 and compete with seven other 841's for channel use? Should it reside on a dedicated 844 with its faster transfer rate, but compete with two other 844's and take up only a small portion of

available disk space (with the remaining space on the pack wasted)? Or should it reside on a 6638 with its dedicated channel, but yet even more wasted space? 1SA has proved to be an invaluable tool in helping to make the above decision. Full configuration testing using 1SA is still continuing at the time of this writing.

7. Finally, 1SA has been a useful tool in the design of a representative benchmark. In benchmarking the SCOPE 3.4 operating system, a one-hour job mix was desired whose resource requirements matched as closely as possible those of a "typical" one-hour period of everyday operation. 1SA data averaged over a random two-week period was used as a "standard". Next, a random mix of jobs was obtained from the input queue and run with 1SA monitoring the system's performance. Monitoring data from this mix was then compared with the "standard". Noting that the random mix was too CPU bound, several synthetic I-O bound jobs were included. This new mix was run and its 1SA data was compared with the "standard". This iteration process continued until a mix was obtained whose 1SA data matched as closely as possible that of the "standard". This final mix was then declared a "representative benchmark" and was used in the benchmarking studies.

V. Future Plans

As the monitoring effort at NWL is a continuing one, enhancements to the monitoring tools are constantly being made. Future plans call for: full-scale hardware monitoring; implementation of a new, more flexible version of SPY; provisions for graphical output from the 1SA data analyzer; a parametric study of system resources based on data collected from 1SA; dynamic interaction between the various monitors and the system resource allocators. A combination of the latter two efforts would effectively eliminate the human from the data-collection $\rightarrow$ human-interpretation $\rightarrow$ system-modification sequence. The performance monitors, observing the state of the machine at time t, could appropriately set various system parameters in order to optimize the allocation of resources at time $t + \Delta t$. An effort is currently underway to employ this technique in order to balance scratch allocation among the three different types of mass storage devices.

VI. Conclusions

This paper summarized the monitoring effort at the Naval Weapons Laboratory, the pre-monitoring analysis required for this effort, and its resultant benefits and future plans. Three software monitoring tools were discussed: a dayfile analysis program, the SPY program, and the 1SA monitor. In addition, several hardware monitoring configurations were proposed. Of all the monitoring tools used to date, the 1SA software monitor was shown to be the most rewarding. Several concrete benefits directly attributable to the use of 1SA were discussed.

Results of the monitoring effort at NWL show that the previously stated goals have been achieved and that, for a minimum investment in man-effort, a computer system as complex as the CDC 6700 can be reasonably measured with existing monitoring tools. In addition, the future incorporation of feedback from the monitors to the operating system should make the system even more responsive to the resource requirements of its dynamically changing workload. A machine capable of "learning" from its own internal monitors would indeed be an important innovation.

FEDSIM STATUS REPORT

Michael F. Morris and Philip J. Kiviat

FEDSIM

FEDSIM

FEDERAL COMPUTER PERFORMANCE

EVALUATION AND SIMULATION CENTER

FEDSIM is the Data Automation Agency's newest and, by far, smallest center. Government-wide interest in this center is evidenced by the fact that FEDSIM's full name was changed from the Federal ADP Simulation Center to the Federal Computer Performance Evaluation and Simulation Center--a change made in response to a request from Congressman Jack Brooks to make FEDSIM's name better fit its mission.

MISSION

Provide reimbursable technical assistance, support, and services throughout the Federal Government for simulation, analysis, and performance evaluation of computer systems.

FEDSIM is unusual in many respects. It provides consultant services throughout the Federal Government to improve the performance of computer systems--both existing and proposed--on a fully cost-recoverable basis. FEDSIM has no budgetary impact on the USAF. All services provided are charged for at a rate that enables FEDSIM to meet its payroll, rent, utility, and contracted costs. FEDSIM's mission addresses a pressing need within the government computer community. This need has been so well recognized by computer managers that FEDSIM is now in an "oversold" condition based almost entirely on customer demands without a "marketing" effort in any formal sense.

FEDSIM's goals and the criteria under which it operates are aimed specifically at fulfilling its mission.

GOALS AND CRITERIA

* Minimize cost of procuring/utilizing computer systems. Maximize productivity of computer systems--efficiency.
* Act on request of user agencies. Maintain confidential FEDSIM-Customer relationship.
* Provide responsive service.
* Cost recovery.
* Provide centralized information/advisory service.

In projects involving proposed systems, FEDSIM strives to make the cost of both acquiring and of using the new computer as low as possible, while meeting all system requirements. This is done through detailed design testing using simulation and mathematical modeling to examine the system as proposed, and the system as it might be improved--before committing resources to creation of any system. Examination of existing systems involves use of devices and techniques that make the fit of resources and demands clear, so that alternate methods may be proposed and tested that will enable the installed equipment to accomplish more work in the same time or the same work in less time.

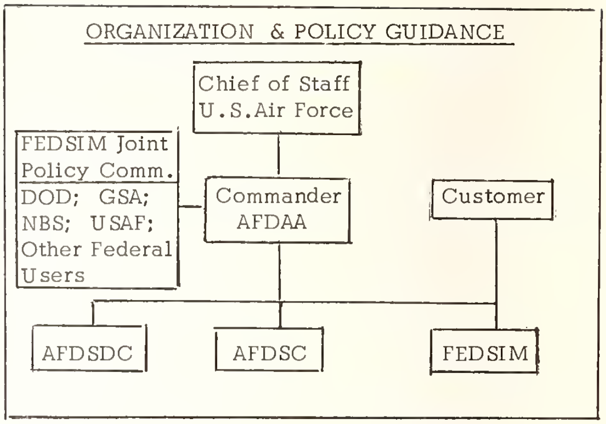
FEDSIM responds to requests of government computer users; FEDSIM cannot go into an installation without an invitation. To insure that FEDSIM will help and not hurt its customers, a confidential relationship is maintained. This encourages the customer to give FEDSIM analysts all of the information about their problem(s) without fear that any practices will be "reported" to higher authorities. As reports prepared are the property of the customer, they are not released by FEDSIM without prior written customer approval.

Because of the nature of FEDSIM's mission, customers usually have a significant problem before they call for FEDSIM assistance. The FEDSIM goal has been to respond to every call as quickly as possible. To do this without an extremely large staff requires that FEDSIM maintain contracts with private companies with specialized computer performance evaluation capabilities that can provide analyst support to meet peak demands. Costs of this support are passed on to customers at the prices paid by FEDSIM. Overhead costs are recovered out of the hourly rate charged for FEDSIM analysts. This insures that no government agency pays more for contractor support through FEDSIM than this same support would cost if purchased commercially. Also included in FEDSIM's overhead is the cost of providing information and advice to Federal agencies on the availability, price, and usefulness of particular performance evaluation techniques and products.

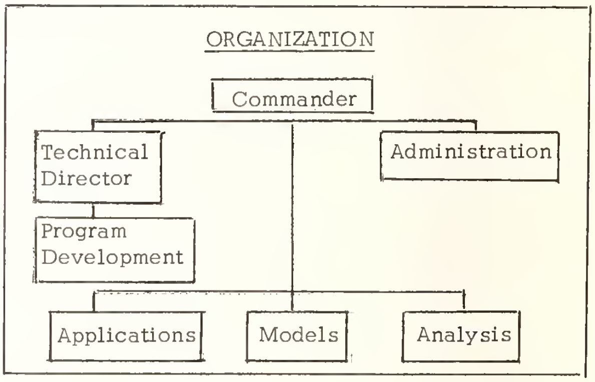
MILESTONES

Sept. 1970	-	GSA requests USAF to act as as Executive Agent.
May 1971	-	USAF agrees.
29 Feb. 1972	-	FEDSIM activated.
1 July 1972	-	Initial operational capability.

This availability of technical expertise on an "as needed" basis for improving the use of government computer systems was one of the reasons behind the General Services Administration's effort to establish FEDSIM and other Federal Data Processing Centers. GSA recognized the need for special skills and experience necessary to provide computer performance evaluation services to all agencies of the government. Because of the record of achievement in this specialty within the Air Force, GSA asked that the Air Force operate FEDSIM. After examining the impact of this unusual offer made in late 1970, the Air Staff agreed in mid-1971 to assume the executive agent role for GSA. To insure that FEDSIM had the necessary authority, GSA made a full delegation of procurement authority for CPE products to the USAF for FEDSIM to exercise. The initial FEDSIM cadre started work towards establishing the needed CPE capability in February 1972; its initial operational capability was achieved on 1 July 1972. This is a remarkably short time for the establishment of a new government agency. The earliest record of a suggestion that there be such an agency places the seed of the idea in early 1969. This rapid step from idea to reality suggests that the "time was ripe" for the birth of this type of organization.


ORGANIZATION & POLICY GUIDANCE
Chief of Staff U.S.Air Force
FEDSIM Joint Policy Comm.
DOD; GSA; NBS; USAF; Other Federal Users
Commander AFDAA
Customer
AFDSDC
AFDSC
FEDSIM

Along with the establishment of FEDSIM, the Air Force organized its centralized computer support activities into the Air Force Data Automation Agency. In recognition of FEDSIM's different mission, a Joint Policy Committee with members from Air Force, the Department of Defense, GSA, and the National Bureau of Standards was established. This committee sets FEDSIM policy, approves capital asset acquisitions, and resolves any priority disputes that may arise between FEDSIM users. FEDSIM is allowed to interact directly with its customers so that no intermediate organizational barriers will be encountered that might decrease FEDSIM's responsiveness to customer demands.



Internally, FEDSIM is organized by natural differences in performance evaluation tools. The three operational divisions are each comprised of particular types of specialists: The Applications Division deals with problems that tend to occur at every computer installation--problems that can be handled with package simulators capable of replicating most commercially available computers running typical data processing workloads; the Models Division creates special simulation models for analysis of unusual computer systems or applications; the Analysis Division performs projects that require special data collection techniques and mathematical analysis. The staff positions are few at FEDSIM. The Technical Director, aided by the Program Development Officer, insures that FEDSIM methods and products represent state-of-the-art use of available technology, and that projects undertaken by FEDSIM are within FEDSIM's capability and jurisdiction. Administration performs the usual administrative tasks and the unusual task of insuring that all project-related time is properly billed to FEDSIM customers.

RESOURCES

*Simulators	*Hardware Monitors
COMET (SCERT)	Microsum
CASE	Dynaprobe 7900
	X-Ray
*Simulation Languages	*Software Monitors
ECSS	SUPERMON
SIMSCRIPT 11.5	PROGLOOK
GPSS	CUE
	PPE

*Skilled People
- Analyst
- Programmers
- Technicians
- Model Builders
- Statisticians

FEDSIM's resources include most of the commercially available performance evaluation products. However, FEDSIM's greatest resource is the skilled people that use these products.

PROJECTS COMPLETED FOR:

General Services Administration
Atomic Energy Commission
Department of Commerce
Internal Revenue Service
Social Security Administration
U.S. Navy (Naval Technical Training Center)
Civil Service Commission
Housing and Urban Development
U.S. Postal Service
National Bureau of Standards
Department of Labor
U.S. Air Force (Accounting & Finance Center)

Through careful use of these resources, FEDSIM had completed projects for 12 Federal Departments and agencies by the end of its first year in operation.

PROJECTS CURRENT FOR:

General Services Administration (3)
U.S. Coast Guard
U.S. Air Force (ACDB/DSC)
Joint Technical Support Agency
Department of Interior
Federal Communications Commission
U.S. Navy (Naval Weapons Laboratory)
U.S. Army (Military Personnel Center)
Department of Transportation
Tennessee Valley Authority

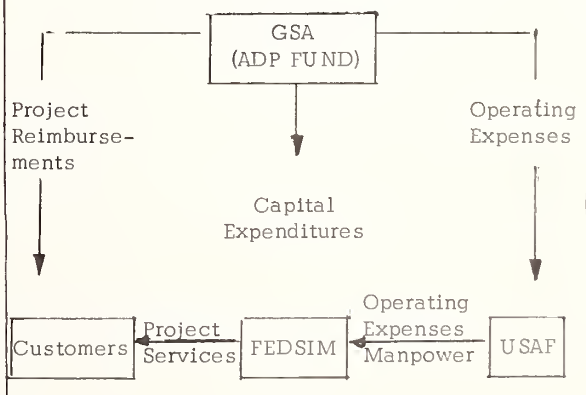
At that time, FEDSIM had computer performance evaluation projects underway for 10 agencies that would result in an income during FY 74 of over $1 million.

TYPICAL PROJECT OBJECTIVES

Technical Evaluation of Proposals
Equipment Reconfiguration
Program Performance Improvement
Facility Study
Computer Sizing/Measurement
Computer System Design
Present and Future Capacity Study

FEDSIM's projects have generally fallen into 7 areas: Technical evaluation of proposals, Equipment reconfiguration, Program performance improvment, Facility study, Computer sizing and measurement, System design, and Studies of existing and future capacity. Hence, FEDSIM is addressing problems of computer performance throughout the life cycle of computer systems from the examination of vendors' proposals to projection of an installation's future computer needs.

AGENCY RELATIONSHIPS



The relationships between FEDSIM and the customer are usually the only visible activities associated with FEDSIM support to a customer. However, the full mechanism of relationships is shown in this diagram. The GSA ADP Fund Serves as the "financial accumulator" in this system. The Air Force furnishes operating manpower and funding and charges these at cost to the GSA Fund on a monthly basis. GSA reimburses the Air Force each quarter. FEDSIM provides project services to its customers and reports all manhour and other costs by project to GSA. GSA bills these services to customer agencies to reimburse the GSA Fund. Capital expenditures for equipment are normally billed directly to GSA for payment from the Fund.

To re-emphasize the simplicity of the FEDSIM-customer relationship, the procedure for obtaining support normally starts by a telephone contact and briefing on FEDSIM's capability and discussion of the customer's problem.

<u>PROCEDURES</u>

▼

Contact

▼

Preliminary Project Plan

▼

Signed Agreement

▼

Project

▼

Deliverables

▼

Follow-up

Based on this contact, FEDSIM works with the customer to prepare a "preliminary project plan" which spells out the problem, service and support, deliverables, and any special arrangements necessary to conduct the project. Once mutual agreement is reached on the plan, it is signed by both FEDSIM and the customer and becomes an "agreement." At this point, the project is begun and periodic reports are made during the project to inform the customer of project status and costs until the work is completed. Deliverables are presented on a schedule that is spelled out in the agreement. FEDSIM has begun a follow-up procedure that involves contacting the customer 30 days after a project is completed to insure that all information has been understood, and to give the customer an opportunity to ask any questions that may have come up relative to the project. A second follow-up approximately 180 days after a project is completed is made to determine the value of the project to the customer, and to find out what actions resulted from FEDSIM's recommendations. This second contact is also intended to encourage any suitable additional projects that may have come up in the interim. These follow-ups serve to insure that the customer got what he expected, and aid FEDSIM's marketing effort.

<u>SERVICES</u>

*Computer performance evaluation consultant services and technical assistance.

*Contractual assistance for purchase, lease, or use of computer performance evaluation products.

*Training in the application of computer performance measurement and evaluation techniques.

To summarize, FEDSIM provides service in computer performance evaluation that is specifically tailored to each individual customer's needs. These services are available for existing installation in the form of technical assistance of consultant services and for the purchase or acquisition of CPE products. FEDSIM also provides training at both the management and technical level to encourage wider use of computer performance evaluation techniques to improve computer usage throughout the Federal Government.

DATA ANALYSIS TECHNIQUES APPLIED TO PERFORMANCE MEASUREMENT DATA

G. P. Learmonth

Naval Postgraduate School

ABSTRACT

The methodology of system performance measurement and evaluation generally requires statistical analysis of data resulting from benchmarking, on-line measurement, or simulations. This paper will survey the pertinent topics of statistical analysis as they relate to system performance measurement and evaluation.

INTRODUCTION

The purpose of this conference is to bring together people with the common interest of measuring and evaluating computer system performance. We have been forced into measuring and evaluating computer system designs by the ever increasing complexity of both hardware and software systems. A quick survey of documented performance studies reveals that nearly every study uses a different approach, usually based on a mixture of experience and intuition. In order to make performance measurement and evaluation a viable discipline, we need to replace experience with theory and intuition with standard research methodology.

The two divergent approaches to computer performance evaluation are the analytic modelling approach and the empirical measurement and evaluation approach. The first approach relies on mathematics, queuing, theory, and probability theory to provide both the theoretical framework and the necessary methodology to perform an analysis. We are concerned here, however, with the second approach. Empirical experiments lack both a theory and a methodology and hence we find ourselves proceeding along tangents making little or no headway.

The theoretical framework of empirical performance evaluations can be defined as soon as our collective experience is filtered and assembled into a cohesive body of knowledge. For a research methodology, we may do as many other sciences, both physical and behavioral, have done; that is, use the existing methods and tools of statistical analysis.

Present day statistical analysis provides two broad areas which may be exploited for performance evaluation. Classical statistical inference techniques require distributional assumptions before data collection takes place. When the data is on hand, systematic application of statistical methods enable the testing of a broad range of hypotheses which may be specified. In the absence of a well-defined theory, this area of statistical methodology is somewhat limited in application to performance measurement problems. Recent developments in the area of exploratory data analysis, however, are extremely promising. In data analysis we are more concerned with indication rather than inference. Data analysis techniques are very flexible and will enable us to incorporate our experience and intuition into an iterative process of analyzing performance data.

This paper intends to survey some pertinent topics in statistical analysis as they might apply to computer system performance measurement and evaluation.

BENCHMARKING: THE CONTROLLED EXPERIMENT

Perhaps the most widely used method of performance measurement and evaluation is benchmarking. This form of experiment is employed to make relative judgments concerning the performance of two or more hardware and/or software configurations. The experiment is accomplished by running either a standard mix of programs or a synthetic mix through each configuration and comparing the results on several criteria.

A critical area in benchmarking is the selection of the job mix. The wrong mix will obviously invalidate any conclusions made. A so-called standard mix consists of a selection of real programs deemed to be representative of the installation's work load. A synthetic job mix involves the use of a highly parameterized program which essentially does nothing constructive but can be tuned to behave in any desired fashion. Any theory of performance evaluation must necessarily contain a precise definition of work load characterization.

Setting aside the problem of proper workload characterization, we may concentrate on the problem of data collection. Pertinent system performance data may be gathered externally via hardware monitors at no system overhead or internally via software monitors with varying degrees of overhead. In most benchmarking experiments, copious data is recorded for later analysis. However, this "shotgun" approach to data collection is generally wasteful and unnecessary. In general, one carefully chosen measure will suffice to answer any single question, for example, throughput, elapsed time, number of page faults, response time, etc.

Typically, a benchmark experiment aims to measure and assess system performance under varying system parameters. These system parameters are controllable by the experimenter and are set to the desired levels of comparison. After all combinations of system parameter settings are subjected to the benchmark, the experimenter must make conclusions regarding the relative performance of the system under these varying conditions.

The description of benchmarking given above is conducive to formal statistical analysis under the category of the Design of Experiments and Analysis of Variance. The properly designed statistical experiment encompasses the proper choice of measured, or dependent variable; the settings of levels of the controlled, or independent variables; the control of external influences; the analysis of the remaining variance; and the testing of the hypotheses under consideration.

The measured variable should be the one which will be directly affected by the choice of independent variables. Similarly, the independent variables should be set at levels which will enable the effect of alteration to be readily determined. Control of external variation should be accounted for in the experimental design in order to balance out these effects which may unknowingly influence the results. For example, consider the problem of determining the effect of main store size and page replacement algorithm on the page replacement rate. Three different benchmark job mixes are to be used. One mix represents, say, batch jobs running in the background, another represents interactive user tasks, and the third, a mixture of both batch and interactive tasks. The controlled variables are main store size and page replacement algorithm. The three different job mixes will influence the response variable, namely, the page replacement rate. However, for this experiment we are not concerned with this effect. In order to compensate for the extraneous variance induced by the job mixes, the following design is adopted.

Assuming three levels of main store, 100, 150, and 200 pages and three replacement algorithms, first-in first-out (FIFO), least recently used (LRU), and a predetermined optimal (BEST), we have nine combinations. We establish the design as follows:

	FIFO	LRU	BEST	
100	J1	J2	J3	J1 = job mix 1
150	J2	J3	J1	J2 = job mix 2
200	J3	J1	J2	J3 = job mix 3

Rather than submit each job stream to each combination of variables thereby performing 27 experiments, we assign each mix once and only once to each row and column. In this way each level of independent variable is subjected to all three job mixes and by virtue of the arrangement, any external influence of the job mix has been balanced-out over the design.

Analysis of variance techniques are available to compute the necessary test statistics. With these statistics the effect of different store sizes may be tested independently of replacement algorithm. Similarly, replacement algorithm may be tested independently of store size. Lastly, any interaction effect between store size and replacement algorithm may also be tested.

Tests on the design of experiments and analysis of variance are available and go into greater detail than has been done here. See, for example, Kirk and Cox. For a very thorough designed experiment in performance analysis see Tsao, Comeau, and Margolin and Tsao and Margolin.

This classical statistical inference technique can be seen to be quite useful. It suffers, however, from the lack of an underlying theory in performance evaluation, particularly in workload characterization and in the necessary distributional assumptions associated with the response variable. However, in the data analysis sense, where the experiment might lack inferential value, it certainly provides great indicative value. The results of an analysis as described above may turn out to be inconclusive, but most certainly they will indicate the direction in which further analysis should be taken. By systematic and orderly design of experiments, much unnecessary and fruitless labor can be avoided.

ON-LINE MEASUREMENT

Benchmarking is widely used but in many instances an installation cannot afford the time and resources to perform stand-alone experiments like those described above. An equally popular measurement technique involves the on-line collection of system data through either hardware or software monitors. Vast amounts of data are collected and stored on tape or disk during normal operating hours.

The stored data is usually treated in two different ways. First, data is edited and reduced for simple description and presentation. Most everyone has seen tabulated data from computer centers listing the number of batch jobs, number of terminal sessions, breakdown of programs by compiler, etc. Elementary statistics such as means, standard deviations, and ranges usually accompany these reports.

It is sometimes desired to assess the impact of major system changes such as operating system releases, addition of hardware, or the rewriting of a major application program. Statistical techniques are available to analyze this impact when sufficient data has been gathered after the change has been made. Classical statistical methods enable the comparison of means and variances to test whether changes in selected variables are significant.

An example of this kind of measurement analysis is given in Margolin, Parmalee, and Schatzoff. Selected on-line measurements automatically recorded by a software monitor within the CP/67 operating system were used to assess the impact on supervisor time of changes to the free storage management algorithm. The authors were careful to design their experiment according to the methods outlined in the preceding section. Their experiment consisted of changing the free storage management algorithm on successive days. The system was not altered in any other way during the experiment and normal user activity took place. To avoid a possible source of extraneous variation, the two-week experiment was designed to have different versions of the free storage management algorithm randomized over the days of the week. It was known that user activity varied over the days of the week with low usage on Fridays and weekends.

While a formal analysis of variance was not performed, the authors did rely on classical as well as data analysis techniques to guide their research effort.

A second way of treating collected on-line measurements is to use the data to uncover interrelationships among system parameters. In this way, the impact of changes in the system may be predicted based on knowledge of these interrelationships.

A statistical method which can be applied in this area of evaluation is regression analysis. As an inferential tool, regression analysis does impose certain assumptions beforehand. However, as in the statistically designed experiment, these prior assumptions may not be strictly met, but we still use regression as an indicative tool.

In general, regression analysis attempts to form a functional relationship between a response, or dependent variable and a set of one or more independent variables. The function is most often taken to be a linear one, thereby keeping the mathematics relatively straight-forward.

In the field of performance measurement, the work of Bard at the IBM Cambridge Scientific Center is the most noteworthy application of regression analysis. In his paper Bard specifies all of the measurements taken by the CP/67 System at the Scientific Center. In four separate sections, he presents the results of four regression analyses. The first application is an attempt to relate system overhead (CPTIME) to other measured variables in the system. The second analysis expresses throughput as a function of the number of active users. The third analysis is concerned with the saturation of the system. Having constructed a regression function for throughput as a measure of overall system performance, it was possible to define the points at which an increase in one variable could only be made at the expense of some other independent variable. The last analysis of his paper is a regression study of the free storage management data previously analyzed by Margolin, Parmalee, and Schatzoff.

While these are only a few examples of the analysis of data measured on-line, the potential usefulness of statistical methods and data analysis are evident.

SIMULATION

Performance measurement and evaluation studies which involve simulation of the computer system are generally undertaken either (1) because analytic models are too complex, or (2) because the system under investigation is not yet operational.

Analytic modelling requires a great number of factors to be expressed and their interrelationships defined in a concise mathematical form. Often this is impossible or simply too complicated and simulations are built which try to mimic the system. Simulations are subject to many possible difficulties concerning the level of detail which will be incorporated into the model. Additionally, the job mix which is run through the simulation must also be properly characterized to cause the model to behave as the real system would.

The lack of proper theory as to how one builds a model and how to characterize workload constitute the difficulties with simulations. For the reasons stated above, however, simulation is often the only recourse available to derive estimates of system performance.

But for the lack of realism, performance measurement and evaluation studies are conducted using the simulation model much as they would be conducted on a real system. The arguments advanced in the two preceding sections for the use of statistical methods and data analysis techniques are equally valid when analyzing the output from a simulation of a computer system.

CONCLUSIONS

We began this paper by pointing out that computer performance measurement and evaluation lacked an underlying theory and research methodology. It was not the aim of the paper to propose a theory, but to call attention to an existing research methodology which has found wide spread use in other disciplines.

It is hoped that convincing arguments have been made to support the assertion that performance measurement and evaluation may find great potential in the application of the statistical and data analytic methods. We feel that by its inherent flexibility, data analysis is perhaps the promising tool.

When adequate theory is available, the inferential methods of classical statistical analysis may be used to test whether what has been observed conforms to that theory.

REFERENCES

Bard, Y., "CP/67 Measurement and Analysis: Overhead and Throughput," in Workshop on System Performance Evaluation, New York, Association for Computing Machinery, March 1971.

Cox, D. R., Planning of Experiments, New York; John Wiley & Sons, Inc., 1958.

Kirk, R. E., Experimental Design Procedures for the Behavioral Sciences, Belmont, California, Brooks/Cole Publishing Company, 1968.

Margolin, B. H., R. P. Parmalee, and M. Schatzoff, "Analysis of Free Storage Algorithms" in Statistical Computer Performance Evaluation, Walter Frieberger (Ed.), New York, Academic Press, 1972.

Tsao, R. H., L. W. Comeau, and B. H. Margolin, "A Multi-factor Paging Experiment· I The Experiment and Conclusions," in Statistical Computer Performance Evaluation, Walter Frieberger (Ed.), New York, Academic Press, 1972.

Tsao, R. H., and B. H. Margolin, "A Multi-factor Paging Experiment: II Statistical Methodology" in Statistical Computer Performance Evaluation, Walter Freiberger (Ed.), New York, Academic Press, 1972.

A SIMULATION MODEL OF AN AUTODIN AUTOMATIC SWITCHING CENTER COMMUNICATIONS DATA PROCESSOR

LCDR Robert B. McManus

Naval Postgraduate School

I. INTRODUCTION

As part of my thesis for a Master's Degree at the Naval Postgraduate School, with the assistance of Professor Norman Schneidewind, my advisor, I developed a simulation model of the AUTODIN Automatic Switching Center (ASC) Communications Data Processor (CDP) located at McClellan Air Force Base, California.

This paper presents a brief network description focusing on the CDP, in order to give an idea of the operation of the AUTODIN as a whole and how the CDP fits into the overall picture. This network description will be followed by a discussion of the simulation program construction, including the more important restrictions required. Next will be highlighted some of the anticipated uses of this and like simulation models. Finally, some conclusions will be drawn concerning the AUTODIN as a result of experiments conducted using the model.

II. BACKGROUND

A. General Network Description

The Automatic Digital Network (AUTODIN) is a switched network, which along with the Automatic Voice Network (AUTOVON) constitutes a major portion of the worldwide Defense Communications System (DCS). The AUTODIN processes 95% of the Department of Defense record traffic, which amounts to about one million messages a day. It currently costs about $80 million a year to operate this system for over 1300 subscribers located throughout the world.

The AUTODIN is made up of Automatic Switching Centers; ASC tributaries, which comprise the 1300 subscribers; and the interconnecting circuits, trunks, transmission links, and terminal facilities.

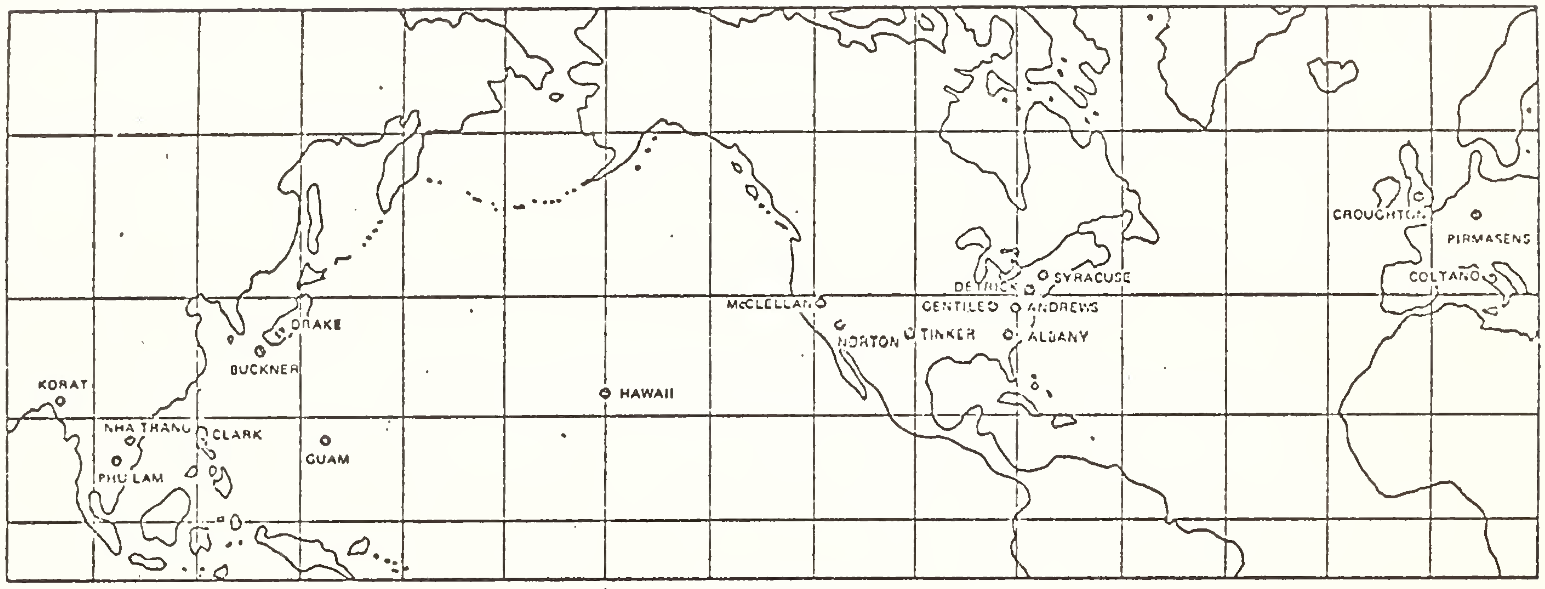
The ASC's are the interconnected message and circuit switching points of the AUTODIN, each originally designed to accommodate 250 message switching users and 50 circuit switching users. There are eight leased ASC's in the United States and eleven government owned ASC's overseas. The geographic location of each ASC is shown in Figure 1. While viewing this figure, it should be pointed out that each ASC is not connected to every other ASC in the network. Rather, the ASC is connected by trunks to the closest five or six ASC's with relay capabilities through these ASC's to the other units in the network.

B. ASC Major Functions

The four major functions performed by the ASC are: message switching, circuit switching, message processing, and message bookkeeping and protection. The message switching function is the only one to be covered in this paper in that it is by far the most important function performed and is the only one dealt with in the simulation model.

The message switching function involves taking the received message from the various incoming channels of the ASC, converting the code and speed as necessary to conform with the intended outgoing channel, and delivering the message to the proper outgoing channel. The message transmitted by the system conforms to certain specified formats, all messages being composed of a header, text, and ending.

Message switching employs the store-and-forward concept of transmission, in which each message is accumulated in its entirety at each ASC on its route, then routed and retransmitted as outgoing channels become available. Selection of messages for retransmission over available outgoing channels is made on a first-in-first-out (FIFO) basis, according to message precedence.



DCS AUTODIN
SWITCHING CENTERS

LOCATION	O & M RESPONSIBILITY	LOCATION	O & M RESPONSIBILITY
1. ALBANY MCSC, GA.	USN	11. CLARK AB. P.I. -	USAF
2. ANDREWS, AFB, MD.	USAF	12. CP DRAKE, JAPAN	USAF
3. FT. DETRICK, MD.	USA	13. NAVCOMSTA FINEGAYAN, GUAM	USN
4. GENTILE AFS, OHIO	USAF	14. NAVCOMSTA WAHIAWA, HAWAII	USN
5. McCLELLAN AFB, CALIF.	USAF	15. KORAT RTAFB, THAILAND	USAF
6. NORTON AFB, CALIF.	USAF	16. NHA TRANG, REP OF VIETNAM	USA
7. HANCOCK FIELD, N.Y.	USN	17. SAIGON, REP OF VIETNAM	USA,
8. TINKER AFB, OKLA.	USAF	18. CROUGHTON RAF STATION, ENGLAND	USAF
	USAF	19. COLTANO SITE, LEGHORN, ITALY	USA
10. FT. BUCKNER, OKINAWA, RYIS	USA	20. PIRMASENS, GERMANY	USA

Figure 1. Geographic Locations of ASCs.

C. ASC Major Equipment

The major functions of the ASC are accomplished by utilizing several special purpose computers, related peripheral devices and necessary communications equipment. A block diagram of the major components of an ASC operating system is provided in Figure 2, which will be a helpful reference in the following discussion of some of the more important pieces of equipment and their operation.

The message arrives at the ASC over incoming channel lines from the ASC tributaries or interconnecting trunks. The message transmission along these circuits is in a carrier form up to the MODEM and a low level D. C. signal thereafter. The MODEMS are also used to convert this analog signal used for transmission in the voice frequency band back to digital signals for ASC internal processing. The purpose of monitoring and measuring instruments is to indicate communication line continuity and signal quality. The cryptographic devices provide communication channel traffic security protection, and the switching facilities connect buffering equipment to the corresponding communication channels.

The Buffer System links the Technical Control facility with the Accumulation and Distribution Units (ADU), provides temporary storage for speed matching and contains the controls necessary to interface the ADU with the communication channels.

The Accumulation and Distribution Unit is a special purpose computer operating with combined wired-in and stored programs. Its purpose is to provide the necessary controls and storage for data both being received from and transferred to the communications channels. Each ASC has three ADU's, with two being on-line and one in stand-by. The ADU holds the messages destined for the CDP in one of three ADU zones, which are the terminus of the input transfer channels and starting point for outgoing transfer channels. These zones have been labeled as High, Medium, and Low Speed Zones. The ADU transfers messages from and to the CDP, translates received message characters into Fieldata code for use in the CDP and stores the lineblocks in the ADU core memory until the complete message is forwarded to the CDP. For outgoing messages, the ADU translates the message from Fieldata to an appropriate code to match the recipient's equipment, and passes the message to the line termination buffer, where it passes out of the ASC.

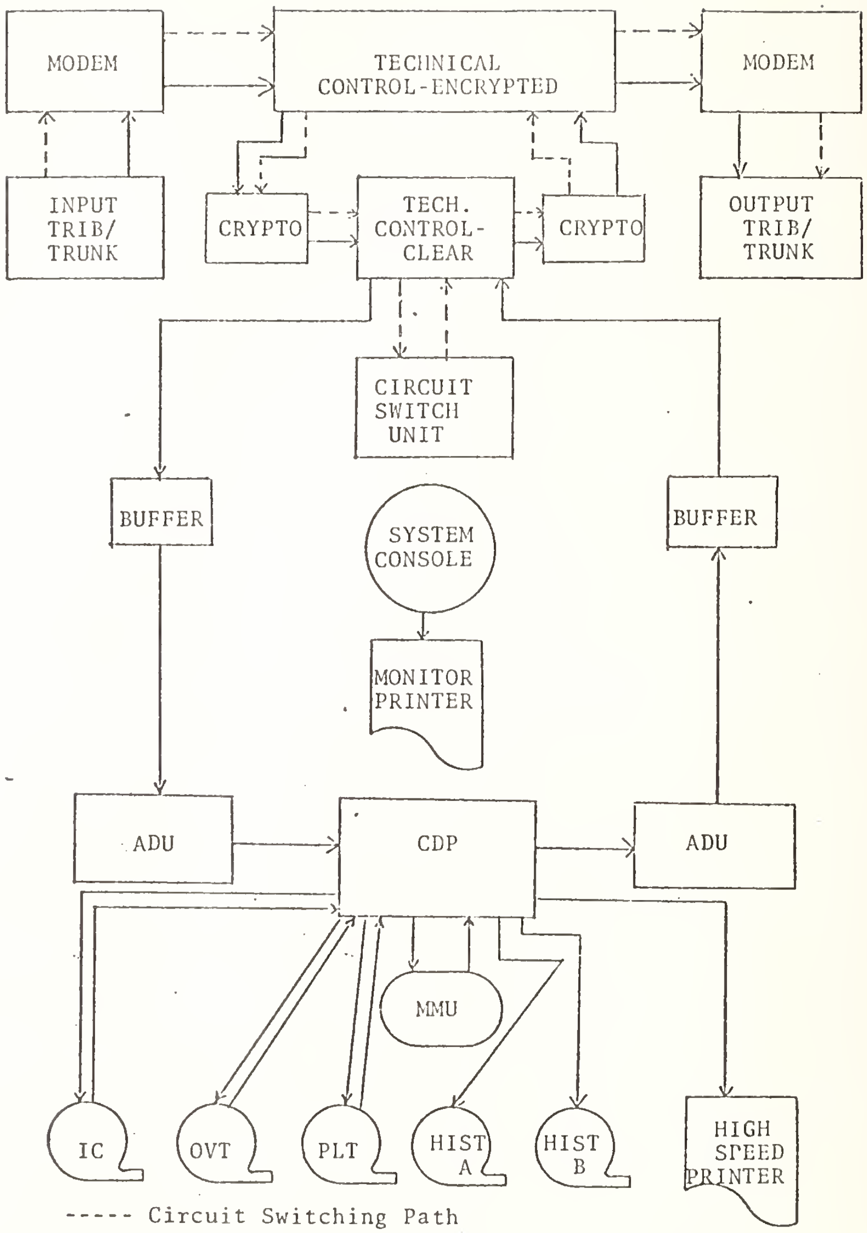
The Communications Data Processor (CDP) is a large-scale digital computer designed especially to perform and control communications functions. The CDP controls the flow of data through the ASC and processes the data according to the AUTODIN program and commands from the system console. The CDP receives the message one lineblock at a time from the ADU. The CDP, upon receipt of the first lineblock, validates the message header and determines proper routing for the message. The lineblocks are then accumulated in memory one lineblock at a time, with an acknowledgment returned after each, until the end of the message is received. Acknowledgment of message receipt is then transmitted to the sender. The message is then linked in the appropriate address queue to await channel assignment and movement on a FIFO basis, subject to message precedence and line availability. There are two CDP's at each ASC with one on-line and the other in stand-by or utilized for off-line work. The CDP is made up of a Basic Processor Unit (BPU), a High-Speed Memory (HSM), a Processor Operator's Console, and a series of Transfer Channels which interface various peripheral devices with the BPU. Two of the most outstanding features of the CDP are its simultaneity and interrupt features. The CDP has the ability to operate several of the peripheral devices asynchronously with internal processing. At the termination of each peripheral device operation, the current CDP program is automatically interrupted, and all register settings are stored. The interrupt feature enables return to this point in the program at a later time without loss of continuity. These features allow the system to initiate peripheral device operation and continue with other tasks with a resulting reduction in message transit times and increase in computer throughput.

Of the many peripheral devices contained in Figure 2, the Intransit Storage Unit or Mass Memory Unit (MMU) and Overflow Tape (OVT) are the two most important to the message switching function. The MMU is the temporary storage area where messages received at the ASC reside until outgoing channels become available and complete retransmission out of the ASC is effected. The MMU consists of disc or drum storage units. The OVT is used to hold messages when "overload" conditions exist, i.e., when the capacity of the MMU storage has been exhausted.

III. SIMULATION PROGRAM CONSTRUCTION

A. Simulation Model Coverage

The portion of the message switching operation



----- Circuit Switching Path

_____ Message Switching Path

Figure 2. ASC Major Components.

covered by the simulation model is contained in Figure 3. Messages are transferred from the three ADU zones via the ADU/CDP transfer channel into the HSM of the CDP. Input processing of the message is performed, which includes interpretation of the destination from the Routing Indicators contained in the heading, acknowledgment of receipt of the message to the sending ASC or tributary, plus other administrative functions.

Upon completion of Input Processing, the message is transferred to Intransit Storage on the MMU via the CDP/MMU transfer channel. Upon availability of the proper channel for the outgoing message, the message is transferred to the HSM again for Output Processing via the MMU/CDP transfer channel. Upon completion of Output Processing, the message is transferred out of the CDP to the appropriate ADU zone via the CDP/ADU transfer channel.

B. Characteristics of the Simulation Model

The simulation model was written in the General Purpose Simulation System (GPSS) language and run on the IBM 360/67 located at the Naval Postgraduate School.

The characteristics of the CDP are treated in this simulation model from the functional point of view. This reduced the level of detail to within manageable limits, enabled the use of the GPSS language, and, in fact, made the simulation effort feasible at all. The individual instructions are not simulated, but times required to perform the various functions are calculated and appropriate delays utilized. If the simulation had been formulated to the level of detail of the individual instruction, the use of GPSS would not have been possible and the CDP simulation program execution would have run slower than the actual instruction execution time. Even with this simplification, one day of real time required over 30 minutes of computer time to execute.

In addition, as was pointed out in the background section, one of the important features of the CDP is its simultaneity of operation and multiprogramming ability. The result of this is that messages are not actually handled serially through the system, but rather, many operations occur simultaneously on different messages throughout the system. The approach taken in the simulation is again a departure from the way things actually happen in the system. Messages are generated and flow serially through the steps of the simulation, with statistics gathered on such items as queues, transit times, storage and facility utilizations, and message sizes. A factor derived from actual system through-put was computed to measure the degree of simultaneity which is present in the CDP. This factor is the ratio of through-put using multiprogramming to through-put using monoprogramming. The simulation times were modified by this factor so that the model does in fact approximately reflect simultaneity of operation. The simultaneity feature of the CDP was incorporated so that mean service times computed by the simulation model would be accurate representations of those actually experienced by the ASC.

In the simulation model, messages were generated at the actual mean arrival rate of messages at the McClellan ASC, calculated over a four-month period. The mean message size was ascertained in the same way.

The characteristics of the actual CDP functions and values assigned are summarized below. See Figure 4, which shows the model components and processing steps, in conjunction with the below description:

1. Messages were generated at the rate of .35 messages per second under normal conditions. In the absence of specific information on the distribution of inter-arrival times, an exponential distribution was assumed.

2. The mean message length was 33 lineblocks. An exponential distribution was assumed. Messages vary in length, but a lineblock is standardized at 80 information carrying characters, plus 4 framing characters. Each character is 8 binary digits in length.

3. The transfer channel between ADU and CDP has a transfer rate of 1240 lineblocks per second.

4. Input processing consists of a mean of 20 thousand computer instructions, exponentially distributed; with from 1 to 77 cycles per instruction, uniformly distributed; and from 1.2 to 2 microseconds of execution time per cycle, uniformly distributed. Input processing is carried out in the HSM which has a capacity of 6461 lineblocks.

5. The transfer between the CDP and MMU has a transfer rate of 4166 lineblocks per second.

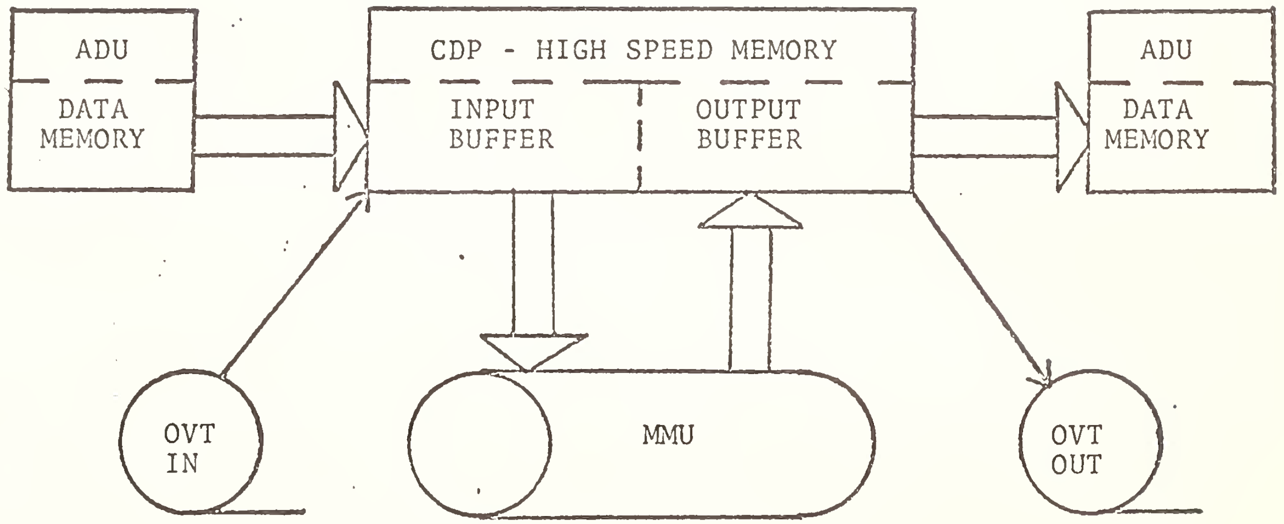


Figure 3. Message Data Flow Through The CDP.

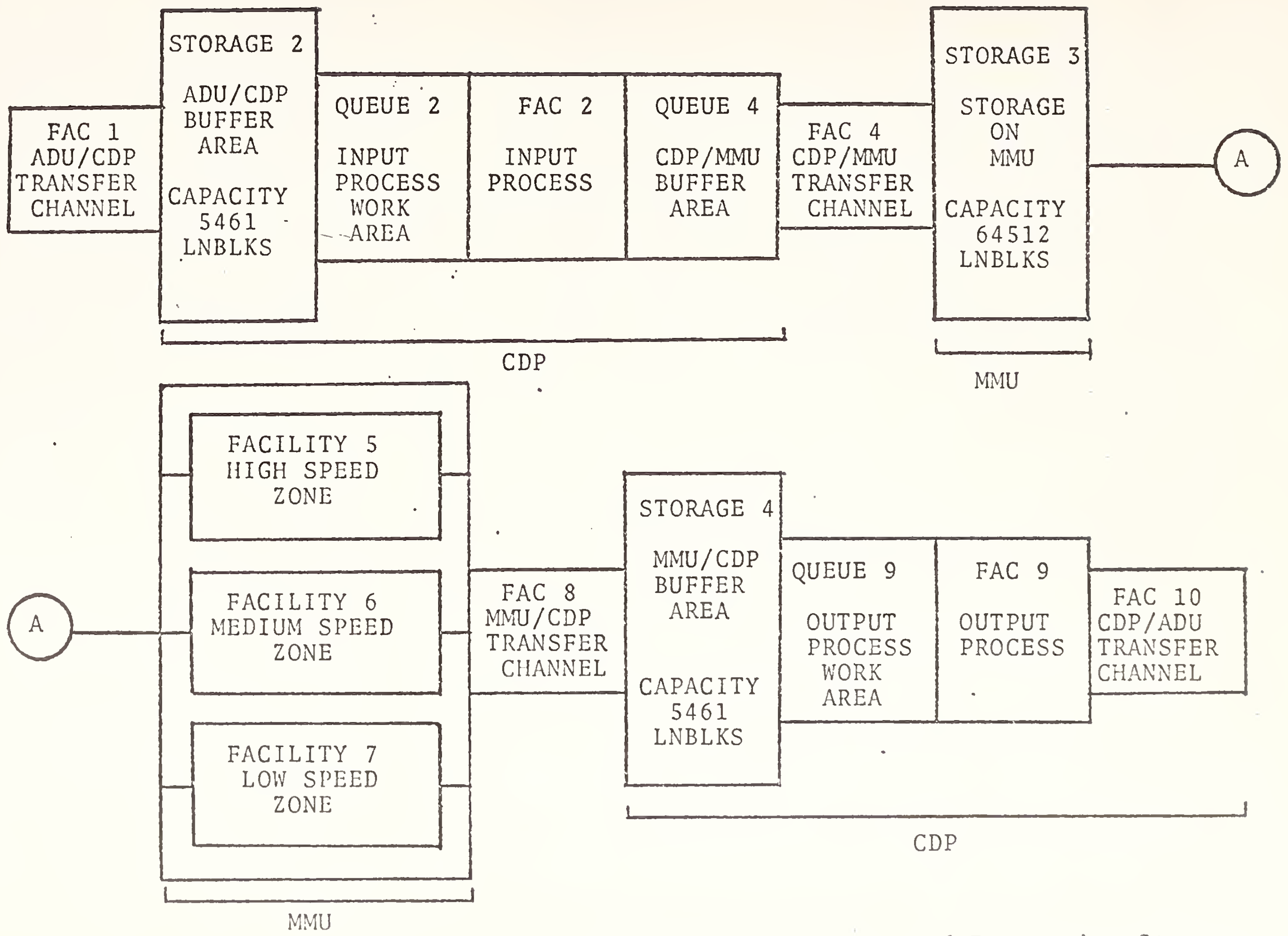


Figure 4. Simulation Model Components And Processing Steps.

6. The MMU has a capacity of 112896 lineblocks, of which 75% is allocated to intransit storage. When 80% of the space allocated to intransit storage is filled, emergency procedures are initiated and low precedence traffic is diverted to the Overflow Tapes. Therefore, the actual capacity of the intransit storage under normal conditions is 64512 lineblocks.

Three user chains were constructed in the MMU to correspond with the three ADU zones. Time delays in each chain and the amount of traffic which occupied each chain was determined by calculating the actual percentage of traffic which was handled by each tributary in that zone. The high speed zone has a capacity of 28 lineblocks and is the terminus of all 4800 and 2400 baud circuits and trunks. A total of 74.3% of the traffic occurs on these lines. The medium speed zone has a capacity of 18 lineblocks and is the terminus of all 1200 and 600 baud lines. A total of 18.6% of the traffic handled by the ASC occurs on these lines. The low speed zone has a capacity of 6 lineblocks and is the terminus of all 300, 150, 75, and 45 baud lines. A total of 7.1% of the traffic occurs on these circuits.

7. Output processing consists of a mean of 10 thousand computer instructions with the same characteristics as for input processing.

With regard to the simulation output, the GPSS automatically provides an output which contains the number of transactions which occupied each block of the simulation, plus a number of performance measurement criteria for items in the simulation. Other items are available, but must be programmed separately. See Figure 5 for a sample of the GPSS output. The more important performance measurement criteria include: mean service times, mean input and output processing times, utilization percentages, mean wait times and average storage contents.

IV. ANTICIPATED USES OF THIS AND LIKE SIMULATIONS

A. Aiding Problem Definition

The analysis necessary to construct the simulation model forces the manager and programmer to state clearly and explicitly his understanding of his system.

B. Isolation of Critical Areas

Systematic testing of the model provides valuable insights into the most sensitive and critical areas of system performance. By using the available data the model can be tested and examined for errors in data values, logic, and functions. Upon satisfactory completion of the model, actual system testing with data values ranging from optimistic to pessimistic can give indications of areas where system performance is weak and where changes are needed or additional analysis is required.

C. Ascertaining Configuration Possibilities

Since the AUTODIN is a relatively complex real-time computer system, there is a variety of design, equipment and configuration possibilities available. Once a model is developed, it can be used to evaluate possible configuration changes. This is vastly superior to actually carrying out the modification of the system and then ascertaining the effect.

V. CONCLUSIONS

A. Experiments Conducted and Their Results

The various experiments conducted using the simulation model were combinations and variations of two basic experiments. First, the rate of inputting transactions was varied. This had the effect in this simulation of changing the system traffic loads into the ASC. Second, the ability of the tributary to receive traffic was varied by modifying the values in the three user chains within the MMU.

From these experiments the following conclusions were drawn:

1. The present CDP is more than adequate to handle even drastic increases in traffic volume.

2. Tributary caused backlogs are going to build up in the CDP and the available storage on the MMU may well be exhausted.

3. The need for an upgrade of the ASC tributaries, on an as-required basis, is indicated.

Figure 5

COMPUTER OUTPUT

```
RELATIVE CLOCK      86400000  ABSOLUTE CLOCK      86400000
BLOCK COUNTS
BLOCK CURRENT      TOTAL    BLOCK CURRENT      TOTAL    BLOCK CURRENT      TOTAL    BLOCK CURRENT      TOTAL    BLOCK CURRENT      TOTAL
  1       0        30129      11      0        30129      21      0        30129      31      0        30129      41      0        22465
  2       0        30129      12      0        30129      22      0        30129      32      0        30129      42      0        22465
  3       0        30129      13      0        30129      23      0        30129      33      0        30129      43      0         7664
  4       0        30129      14      0        30129      24      0        30129      34      0        22465      44      0         7179
  5       0        30129      15      0        30129      25      0        30129      35      0        22465      45      0         7116
  6       0        30129      16      0        30129      26      0        30129      36      0        22465      46      0         7116
  7       0        30129      17      0        30129      27      0        30129      37      0        22465      47      0         7116
  8       0        30129      18      0        30129      28      0        30129      38      0        22465      48      0         7116
  9       0        30129      19      0        30129      29      0        30129      39      0        22465      49      0         7116
 10       0        30129      20      0        30129      30      0        30129      40      0        22465      50      0         7116

BLOCK CURRENT      TOTAL    BLOCK CURRENT      TOTAL    BLOCK CURRENT      TOTAL    BLOCK CURRENT      TOTAL    BLOCK CURRENT      TOTAL
 51       0         7116      61      0          485      71      0        30066      81      0        30066      91      0            0
 52       0         7116      62      0          485      72      0        30066      82      0        30066      92      0           63
 53       0         7116      63      0        30066      73      0        30066      83      0        30066      93      0           63
 54       0          485      64      0        30066      74      0        30066      84      0        30066      94      0          100
 55       0          485      65      0        30066      75      0        30066      85      0        30066      95      0          100
 56       0          485      66      0        30066      76      0        30066      86      0        30066
 57       0          485      67      0        30066      77      0        30066      87      0        30066
 58       0          485      68      0        30066      78      0        30066      88      0        30066
 59       0          485      69      0        30066      79      0        30066      89      0        30066
 60       0          485      70      0        30066      80      0        30066      90      0            0


USER CHAIN        TOTAL         AVERAGE          CURRENT          AVERAGE          MAXIMUM
                 ENTRIES       TIME/TRANS        CONTENTS         CONTENTS         CONTENTS
  HISPD            281           37.654                             .000               2
  MEDSP             63           93.952                             .000               2
  LOSPD              1            8.000                             .000               1


FACILITY        AVERAGE        NUMBER         AVERAGE          SEIZING    PREEMPTING
              UTILIZATION      ENTRIES       TIME/TRAN        TRANS. NO.  TRANS. NO.
     1            .000          30129           1.892
     2            .047          30129         136.669
     4            .000          30129            .282
     5            .008          22465          32.537
     6            .008           7116          98.666
     7            .001            485         186.903
     8            .000          30066            .282
     9            .023          30066          66.698
    10            .000          30066           1.891


STORAGE      CAPACITY      AVERAGE        AVERAGE         ENTRIES        AVERAGE         CURRENT        MAXIMUM
                           CONTENTS     UTILIZATION                     TIME/TRAN        CONTENTS       CONTENTS
     2          5461        1.718          .000          982301         151.135                          453
     3         64512      916.224          .014          982301       80588.125          2207           2479
     4          5461         .788          .000          980094          69.534                          279


CONTENTS OF FULLWORD SAVEVALUES (NON-ZERO)
SAVEVALUE  NR,         VALUE       NR,        VALUE        NR,        VALUE        NR,        VALUE        NR,        VALUE
            2          2207


QUEUE       MAXIMUM      AVERAGE       TOTAL       ZERO       PERCENT        AVERAGE        $AVERAGE        TABLE        CURRENT
            CONTENTS     CONTENTS      ENTRIES     ENTRIES     ZEROS        TIME/TRANS      TIME/TRANS      NUMBER       CONTENTS
   2           4           .003        30129       28744       95.4           9.002          195.848
   4           1           .000        30129       30114       99.9            .000            1.466
   9           3           .000        30066       29262       97.3           2.775          103.803
  $AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES
```

In the final analysis, the simulation program developed and discussed in this paper permits a limited analysis of the ASC and its CDP. As follow-on efforts provide tools for analyzing additional portions of the AUTODIN ASC, more comprehensive analyses can be made. Hopefully, one day all segments will be completed and can be put together so that their interaction can give some indication of overall system performance.

BIBLIOGRAPHY

AUTODIN System and Equipment Configuration, Technical Training Manual 4ALT 29530-1-I-1, Department of Communications Training, Sheppard AFB, Texas, February 1971.

Boehm, B. W., Computer Systems and Analysis Methodology· Studies in Measuring, Evaluating, and Simulating Computer Systems, Rand Corporation Report R-520-NASA September 1970.

Computer Sciences Corporation Report R413681-2-1, DIN Simulator Program Reports, May 1972.

DCA AUTODIN CONUS-System Instruction Manual, RCA/DCS Division, Defense Electronics Products, Camden, N. J., 1 August 1969.

DCS AUTODIN Switching Center and Tributary Operations, Defense Communications Agency Circular 310-D70-30, 10 June 1970.

DCS Traffic Engineering Practices, Defence Communications Agency Circular 300-70-1, May 1971.

Martin, James, Design of Real-Time Computer Systems, Prentice-Hall Inc., Englewood Cliffs, N. J., 1967.

Martin, James, Programming Real-Time Computer Systems, Prentice-Hall Inc., Englewood Cliffs, N. J., 1965.

Martin, James, Telecommunications Network Organization, Prentice Hall Inc., Englewood Cliffs, N. J., 1970.

Nabb, Alan, Captain, USAF, Norton ASC Communications Operating Performance Summary, Norton AFB, California, May 1972.

Norman, Eugene and Lloyd Rice, "Research Report-AUTODIN and its Naval Communications Interface," Management Quarterly, May 1972.

Reddig, Gary A., Captain, USAF, Communications Operating Performance Summary, McClellan AFB, California, 13 October 1972.

Reitman, Julian, Computer Simulation Applications, John Wiley and Sons, Inc., New York, 1971.

Rubin, Murry and Haller, C. E., Communications Switching Systems, Reinhold Publishing Corp., New York, 1966.

Sasieni, Maurice and Yaspan, Arthur and Lawrence, Friedman, Operations Research--Methods and Problems, John Wiley and Sons Inc., New York, 1959.

Schulke, H. A., BGEN/USA, "The DCS Today and How it Operates," Signal, July 1971.

Seaman, P. H., "On Teleprocessing System Design--The Role of Digital Simulation," IBM Systems Journal, Vol. 5, NR 3, 1966.

Study Guide, AUTODIN Switching Center Operations--General Description of the AUTODIN Program, Sheppard AFB, Texas, 15 July 1970.

Wagner, Harvey, Principles of Management Science with Applications to Executive Decisions, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1970.

_______. Personal Interview with R. A. Ondracek, Western Union Maintenance Analyst, ASC McClellan AFB, California, 3 November 1972.

_______. Personal Interview with LCDR Hansen, OINC, Naval Message Center, Naval Postgraduate School, Monterey, California, 18 January 1973.

_______. Personal Interview with Alan Nabb, Captain/USAF, OINC ASC Norton AFB, California, 14 July 1972.

Schriber, Thomas J., General Purpose Simulation System/360: Introductory Concepts and Case Studies, The University of Michigan, Ann Arbor, Michigan, September 1968.

ATTENDEES

EIGHTH MEETING OF

COMPUTER PERFORMANCE EVALUATION USERS GROUP

Marshall D. Abrams
National Bureau of Standards
Technology B216
Washington, D.C. 20234

PFC Edward B. Allen
US Army Computer Systems Command
CSCS-AT STOP H-14
Ft. Belvoir, Va. 22060

Russell M. Anderson
1900 Lyttonsville 1002
Silver Spring, Md. 29010

Philip Balcom
US Army Computer Systems Command
STOP H-6
Ft. Belvoir, Va. 22060

Dr. N. Addison Ball
National Security Agency
Ft. Meade, Md. 20755

Rudie C. Bartel
HQ USAF (ACDC)
Washington, D.C. 20715

Merton J. Batchelder
US Army Computer Systems Command
7719 Bridle Path Lane
McLean, Va. 22101

T. K. Berge
FEDSIM/CC
Washington, D.C. 20022

Bob Betz
Boeing Computer Services
12601 SE 60 St.
Bellevue, Wash. 98006

Raymond S. Blanford
Defense Supply Agency
Cameron Station Rm. 4A-612
Alexandria, Va. 22314

John A. Blue
US Navy ADPESO
Crystal Mall
Washington, D.C. 20376

John J. Bongiovanni
AFDSDC/SYO
Gunter AFB, Al. 36114

Jim N. Bowie
AMC ALMSA
210 North 12th Street
St. Louis, Mo. 63188

N. J. Brown
NAVCOSSACT
Code 10.1 Wash Navy Yd.
Washington, D.C.

Richard A. Castle
US Army Computer Systems Command
Ft. Belvoir, Va. 22060

John F. Cavanaugh
NAVORD SYSCOM CENO
US NAV ORD STA
Indian Head, Md. 20640

Dr. Yen W. Chao
FEDSIM US Air Force
9724 Eldwick Way
Potomac, Md. 20854

Dennis R. Chastain
USGAO
441 G St. NW RM 7131
Washington, D.C. 20548

Leo J. Cohen
Pres., Performance Development Corp.
32 Scotch Rd.
Trenton, N.J. 08540

Dr. Dennis M. Conti
Naval Weapons Lab
Code: KPS
Dahlgren, Va. 22448

L. J. Corio
Western Electric Co.
222 Broadway
New York, N.Y. 10038

Joe M. Dalton
Control Data Corp.
5272 River Road
Washington, D.C. 20016

Jim Dean
Social Security Administration
6401 Security Blvd.
Baltimore, Md. 21235

Donald R. Deese
FEDSIM/AY
HQ USAF
Washington, D.C. 20330

Jules B. Dupeza
DOT
11900 Tallwood Ct.
Potomac, Md. 20854

Gordon Edgecomb
Chemical Abstracts Service
Ohio State University
Columbus, Ohio 43210

Emerson Eitner
Chemical Abstracts Service
Ohio State University
Columbus, Ohio 43210

Richard B. Ensign
FEDSIM
Washington, D.C. 20330

Joseph Escavage
Dept of Defense
9800 Savage Rd. C403
Ft. Meade, Md. 20755

Robert H. Follett
IBM
1401 Fernwood Road
Bethesda, Md. 20034

Dr. K. E. Forry
USACCOM-COMPT-SA
Ft. Huachuca, Ariz. 85635

E. A. Freiburger
Senior Marketing Engineers
3718 Tollgate Terrace
Falls Church, Va. 22041

Robert Gechter
Dept of Interior, CCD
19th & C Sts NW
Washington, D.C. 20242

CPT Jean-Paul Gosselin
DND Computer Centre, Tunney's Pasture
Ottawa, Ontario
Canada KIAOK2

James M. Graves
USAMSSA
Room BD1040 The Pentagon
Washington, D.C. 20330

Donald A. Greene
DSA-DESC-AST
1507 Wilmington Pike
Dayton, Ohio 45444

Lowell R. Greenfield
NASA FRC
Box 273
Edwards, California 93523

Robert A. Grossman
NAVCOSSACT
Bldg. 196 Washington Navy Yard
Washington, D.C. 20374

Vincent C. Guidace
Dept of AF FEDSIM
Washington, D.C. 20330

Col Michael J. Hamberger
DSA, Cameron Station
11207 Leatherwood Dr.
Reston, Va. 22070

I. Trotter Hardy
General Services Administration, CDSG
Rm. 6407 ROB3 7th & D Sts SW
Washington, D.C. 20407

Norman Hardy
PR:SD Room 3137 IRS
1111 Constitution Ave. NW
Washington, D.C. 20007

MAJ W. L. Hatt , CAF
USAF/ESD/MCST-2
L. C. Hanscom Field
Bedford, Mass. 01730

CPT Harry A. Haught
Navy Annex
Arlington, Va.

Herman H. Heberling
Fleet Material Support Office
Mechanicsburg, Pa. 17055

MAJ Andrew Hesser
USMC
HQTRS (ISMDS)
Washington, D.C. 20380

William J. Hiehle
Defense Electronics Supply Çenter
1102 E. Linden Ave.
Miamisburg, Ohio 45342

Dr. Harold J. Highland
Chairman SIGSIM
562 Croydon Road
Elmont, N.Y. 11003

Lt. Donald W. Hodge, Jr.
HQ SAC (ADISE)
Offutt AFB, NE 68113

John N. Hoffman
Boeing Computer Services
955 L'Enfant Plaza North SW
Washington, D.C. 20024

John V. Holberton
GSA
10130 Chapel Road
Rockville, Md. 20854

Marjorie E. Horan
MITRE
1820 Dolley Madison Blvd.
McLean, Va. 22101

Larry G. Hull
NASA-GSFC
Code 531.2 Goddard Space Flight Center
Greenbelt, Md. 20771

George Humphrey
Naval Underwater Systems
Newport Naval Base
Newport, R.I. 02871

Frank S. Hunter
US Army Computer Systems Command
7704 Random Run Lane, Apt. 104
Falls Church, Va. 22042

Jau-Fu Huoang
NAVCOSSACT, Dept of Navy
Bldg. 196, Washington Navy Yard
Washington, D.C.

LT Robert L. James
USAF-AFDSC
AFDSC/GMJ The Pentagon
Washington, D.C. 20330

W. Richard Johnsen
US Army Computer Systems Command
CSCS-ATA-S, Stop H-14
Ft. Belvoir, Va. 22060

Lt. Raymond S. Jozwik
USAMSSA
2500 N. Van Dorn St.
Alexandria, Va. 22302

B. A. Ketchledge
Bell Laboratories
Piscataway, N.Y. 08704

Philip J. Kiviat
FEDSIM
Dept of Air Force
Washington, D.C. 20330

John T. Klaras
Dept of Air Force
Rm 1D118, Pentagon
Washington, D.C. 20330

Ronald W. Korngiebel
Connecticut General Life
Bloomfield, Connecticut 06002

Mitchel Krasny
NTIS
5285 Port Royal Rd.
Springfield, Va. 22151

David W. Lambert
Mitre Corporation
P. O. Box 208
Bedford, Mass 01730

John H. Langsdorf
HQ USAF (ACD)
Washington, D.C. 20330

Don Leavitt
Computer World
797 Washington St.
Newton, Mass. 02160

William J. Letendre
HQ ESD/MCDT Stop 36
Hanscom AFB, Ma. 01730

William P. Levin
Defense Supply Agency
Cameron Station, Rm. 4B587
Alexandria, Va. 22314

Jack R. Lewellyn
HQ NASA
Code TN
Washington, D.C. 20546

Allen LI
Federal Hwy Adm
400 7th St. S.W.
Washington, D.C. 20590

MAJ J. B. Lloyd
AFDSC
The Pentagon
Washington, D.C. 20330

Alex Machina
CENO
P.O. Box 258
Indianhead, Md. 20640

Neil Marshall
Pinkerton Computer Consultants
8810 Walutes Circle
Alexandria, Va. 22309

Sally Mathay
Central Nomix Ofc
NAVORDSTA
Indianhead, Md. 20640

CPT Norm Mejstrik
Air Force Comm Service/L-93
Tinker AFB, Okla 73145

Ann Melanson
USAF
Hanscom Field
Bedford, Mass 01730

CPT Kenneth Melendez
USAF, ESD, L. G. Hanscom Field
ESD/MCI
Bedford, Mass 01730

Anne Louise Moran
Dept of Defense
Ft. Meade C403
Ft. Meade, Md.

Michael F. Morris
FEDSIM
1919 Freedom Lane
Falls Church, Va. 22043

Deanna J. Nelson
DOD Computer Institute
Washington Navy Yard
Washington, D.C. 20374

George R. Olson
Control Data Corp.
8100 34th Ave SO Box O HQNIOU
Minneapolis, Minn. 55440

W. J. O'Malley
Internal Revenue Service
1111 Constitution Ave NW
Washington, D.C.

J. P. O'Rourke
NAVCOSSACT
Washington Navy Yard
Washington, D.C. 20374

Chris Owers
Social Security Administration
6401 Security Blvd.
Baltimore, Md. 21235

Charles W. Pabst
NAVCOSSACT
Washington Navy Yard
Washington, D.C. 20374

Frank J. Pajerski
NASA - Goddard
Code 531 NASA-Goddard SFC
Greenbelt, Md. 20371

Frank V. Parker
US Navy (NAVCOSSACT)
819 Duke St.
Rockville, Md. 20850

Richard D. Pease
IBM Corp.
10401 Fernwood Rd.
Bethesda, Md. 20034

Emery G. Peterson
USATRADOC-DPFO
Fort Leavenworth, Kansas 66027

MAJ Ernest F. Philipp
USAF/SAC
7910 Greene Cir.
Omaha, Ne 68147

Charles Pisano
Fleet Material Support Office
Mechanicsburg, Pa. 17055

Terry W. Potter
Bell Telephone Laboratories
Box 2020 Rm. 4A1105
New Brunswick, N.J. 08903

Marvin D. Raines
US Army Computer Systems Command
513 Shelfar Pl.
Oxon Hill, Md. 20022

Mr. C. D. Ritter
US Geological Survey
4615 Gramlee Circle
Fairfax, Va. 22030

G. F. Rizzardi
Dept of Air Force
AFDSC/SF, Pentagon
Washington, D.C. 20330

E. E. Seitz
US Army R&D
3205 Plantation Pkwy.
Fairfax, Va. 22030

William Selfridge
US Army Computer Systems Command
Stop H-6
Ft. Belvoir, Va. 22060

A. C. Shetler
The Rand Corporation
1700 Main Street
Santa Monica, Wash. 90406

Ronald L. Shores
Atomic Energy Commission
Washington, D.C. 20545

Eileen Silo
Dept of Defense
9800 Savage Rd.
Ft. Meade, Md. 20755

George J. Skurna
Defense Elect. Supply Ctr.
1507 Wilmington Pike
Dayton, Ohio 45444

Charles L. Smith
The Mitre Corporation
Westgate Research Park
McLean, Va. 22101

George M. Sokol
Deputy for Engineering
US Army Computer Systems Command
Ft. Belvoir, Va. 22060

David V. Sommer
NSRDC
Annapolis, Md. 21401

John W. H. Spencer
Bureau of the Census
Statistical Research Division
Washington, D.C. 20233

James Sprung
FEDSIM
5240 Port Royal Rd.
Springfield, Va. 22151

Don W. Stone
TESDATA Systems Corp.
235 Wayne Street
Waltham, Mass. 02154

Hal Stout
COMRESS
2 Research Ct.
Rockville, Md. 20850

Herbert D. Strong
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, Calif. 91011

COL. H.J. Suskin
Defense Electronics Supply Center
Wilmington Pike
Dayton, Ohio

Marvin Sendrow
Fed. Home Loan Bank Board
101 Indiana Ave. N.W.
Washington, D.C. 20552

Kay Shirey
US Army MILPERCEN
DAPC-PSB-B Hoffman II Bldg.
200 Stovall St.
Alexandria, Va. 22332

Nora M. Taylor
Naval Ship R&D Center
Code 189.1
Bethesda, Md. 20034

Harold J. Timmons
DSA, Data Systems Automation Office
c/o Defense Construction Supply Ctr.
Columbus, Ohio 43215

Terry L. Traylor
NAVCOSSACT Bldg 196, Code 101
Washington Navy Yard
Washington, D.C.

John B. Trippe
OA-914, The Pentagon
Washington, D.C. 20330

Ray G. Tudor
SAC HQ (USAF)
814 Matthies Dr.
Papillion, Ne 68046

Ralph R. Turner
US Army Tradoc - DPFO
Bldg 136
Ft. Leavenworth, Kansas 66027

Daniel M. Venese
The Mitre Corporation
Westgate Research Park
McLean, Va. 22101

Daniel A. Verbois
USAMC - ALMSA
P.O. Box 1578 AMXAL-TL
St. Louis, Mo. 63188

H. C. Walder
Western Electric Co.
222 Broadway
New York, N.Y. 10038

F. C. Warther
TESDATA
7900 Westpark Dr.
McLean, Va. 22101

Michael E. Wentworth, Jr.
Social Security Administration
6401 Security Blvd.
Baltimore, Md. 21235

David P. Wheelwright
GSA/ADTS/CDSD
7th & D Sts. S.W.
Washington, D.C. 20407

Brian R. Young
Informatics, Inc.
1502 Kennedy Plaza
Bellevue, Ne. 68005

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET**	1. PUBLICATION OR REPORT NO. NBS Special Publ. 401	2. Gov't Accession No.	3. Recipient's Accession No.
4. TITLE AND SUBTITLE Computer Performance Evaluation			5. Publication Date September 1974
			6. Performing Organization Code
7. AUTHOR(S) Various: Edited by Dr. Harold Joseph Highland, State University Agricultural and Technical College: N. Y.			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)* FIPSCAC, National Bureau of Standards, Room B264, Building 225, Washington, D.C., 20234			13. Type of Report & Period Covered final
			14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

Library of Congress Catalog Number: 74-13113

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 Eighth Meeting of the Computer Performance Evaluation Users Group [CPEUG], sponsored by the United States Army Computer Systems Command and the National Bureau of Standards, was held December 4-7, 1973 at NBS, Gaithersburg. The program chairman for this meeting was Merton J. Batchelder of the US Army Computer Systems Command at Fort Belvoir VA 22060 {CSCS-ATA Stop H-14}.
About 150 attendees at this meeting heard the 17 papers presented on computer performance, evaluation and measurement. Among the papers presented were those dealing with hardware and software monitors, workload definition and benchmarking, a report of FIPS Task Force 13, computer scheduling and evaluation in time-sharing as well as MVT environment, human factors in performance analysis, dollar effectiveness in evaluation, simulation techniques in hardware allocation, a FEDSIM status report as well as other related topics.
These proceedings represent a major source in the limited literature on computer performance, evaluation and measurement.

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)*

Computer evaluation; computer performance; computer scheduling; hardware monitors; simulation of computer systems; software monitors; systems design and evaluation; time-sharing systems evaluation

18. AVAILABILITY	19. SECURITY CLASS (THIS REPORT)	21. NO. OF PAGES
[✓] Unlimited	UNCLASSIFIED	155 pages
☐ For Official Distribution. Do Not Release to NTIS	20. SECURITY CLASS (THIS PAGE)	22. Price
[✓] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13. 10:401	UNCLASSIFIED	$1.80
☐ Order From National Technical Information Service (NTIS) Springfield, Virginia 22151		