NBS SPECIAL PUBLICATION 406

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

Computer Performance Evaluation: Report of the 1973 NBS/ACM Workshop

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: Report of the 1973 NBS/ACM Workshop

Edited by:

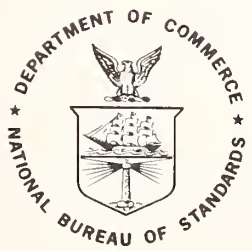
Thomas E. Bell and Barry W. Boehm

TRW Systems
Redondo Beach, California 90278

and

S. Jeffery

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



U.S. DEPARTMENT OF COMMERCE, Rogers C. B. Morton, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Acting Director*

Issued September 1975

Library of Congress Cataloging in Publication Data
Main entry under title:

Computer Performance Evaluation.

(NBS Special Publication; 406)
Supt. of Docs. No.: C13.10:406
1. Electronic Digital Computers—Evaluation. I. Bell, Thomas E. II. Boehm, Barry W. III. Jeffery, Seymour, 1922- . IV. United States. National Bureau of Standards. V. Association for Computing Machinery. VI. Series: United States. National Bureau of Standards. Special Publication; 406.
QC100.U57 No. 406 [QA76.5] 389'.08s [621.3819'58'2]
75-619080

National Bureau of Standards Special Publication 406

Nat. Bur. Stand. (U.S.), Spec. Publ. 406, 180 pages (Sept. 1975)

CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1975

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402 - Price $2.45 (paper cover)
Stock Number 003-003-01393 (Order by SD Catalog No. C13.10:406).

Abstract

An ACM/NBS Workshop on Computer Performance Evaluation (CPE) was held in San Diego, Calif. in March 1973. The papers, workshop discussions, conclusions and recommendations presented in this volume address specific problems in making computer performance evaluation a commonplace and productive practice.

While several of the conclusions indicate that improvements are needed in performance analysis tools, another suggests that improved application of CPE could be achieved by better documentation of analysis field to develop its full potential. Particapants noted that the common emphasis on data collection or modeling, to the exclusion of considering objectives, often seriously degrades the value of performance analysis.

Key words: Computer architecture; computer performance evaluation; installation management; measurement; modeling; monitoring; operating systems; performance objectives.

Foreword

The Institute for Computer Sciences and Technology at the National Bureau of Standards, U.S. Department of Commerce, and the Association for Computing Machinery, the nation's largest technical society for computing professionals, have been jointly sponsoring a series of workshops and action conferences on national issues. These workshops were designed to bring together the best talents in the country in their respective areas to establish a consensus on (1) current state of the art, (2) additional action required, and (3) where the responsibility for such action lies.

Computer Performance Evaluation (CPE) was selected as a subject of an ACM/NBS Workshop* because of the significant leverage CPE activities can have on computer usage. Under the right conditions, CPE can be a major force for improving the efficiency and effectiveness of computer and related human operations.

However, it is important to beware of opportunities to misuse CPE techniques. Collecting vast amounts of data does not guarantee system improvement. Even when one can make an improvement to one system parameter, this may have a degrading effect on another parameter. Another serious problem arises when the results of modeling a system are impossible to validate.

It is hoped that the publication of these proceedings will be of value to specialized investigators in the particular area discussed, as well as those actively engaged in attacking the broad problem of bringing CPE techniques into productive practice.

Thomas E. Bell
TRW Systems Group

Barry W. Boehm
TRW Systems Group

S. Jeffrey
National Bureau of Standards

* The initial planning was supported by a grant from the National Science Foundation.

Contents

EXECUTIVE SUMMARY

Performance Evaluation Workshop

Computer Performance Evaluation (CPE) was selected as a subject of an ACM/NBS Workshop because of the significant leverage CPE activities can have on computer usage. Under the right conditions, CPE can be a major force for improving the efficiency and effectiveness of computer and related human operations.

In addition to the potential savings, there are also significant opportunities to misuse CPE techniques. Several Workshop participants were familiar with situations in which the only result of a $100,000-range CPE effort was a stack of unanalyzable data tapes. Other large efforts have foundered on impossible-to-validate model results. Others have been successful in achieving hardware efficiencies, but have thereby introduced more damaging inefficiencies in the organization's overall productivity because of effects like increased job turnaround times, frequent changes in procedures and user interfaces, and reduced flexibility to cope with emergencies.

Gains in efficiency and overall productivity will not occur if knowledge remains concentrated in a few segments of the industry, each with a slightly different approach and nomenclature. The workshop was organized to expose these differences, to reconcile them if possible, and to achieve a consensus of the state of the art and its future.

A number of conclusions regarding the state of the art can be drawn from the discussions described in this report. Some of these conclusions were explicitly discussed but others were implicit in the attendees' statements throughout the workshop. The primary conclusions determined at the workshop are given below, with references to Chapter I of the text, which serves as a workshop summary. (In turn, Chapter I contains references to the more detailed material in subsequent chapters which was prepared for the workshop by the participants.)

1. Productivity increases averaging 15 percent to 30 percent are generally achievable on third generation computer equipment. However, only increases resulting in improved value or decreased cost have utility to the organization. (Sections I.A.1 and I.A.6.)

2. The current state of the art could support far more productivity increases than are realized, but inadequate motivation and insufficient education limit its application. This situation is aggravated by the lack of a handbook to apply the current knowledge. (Sections I.D.7, I.E.6 and I.E.7.)

3. Improvements in application programs and direct user provisions offer the most potential for productivity gains within the next three years as well as over the long term. Overcentralization of cost-benefit evaluation (e.g., setting priorities or determining maximum allowable core requirements) can adversely affect users' motivations and abilities to achieve these gains. (Sections I.B.3 and I.B.4.)

4. Measurement does not constitute performance evaluation. Evaluation takes place with respect to the objectives and goals of an organization and usually includes some measurement. (Section I.B.5.)

5. Employing appropriate methodological techniques is critical if performance measurement or modeling is to be successful. Picking a measurement tool, collecting some data, and then attempting an analysis will seldom suffice as an evaluation procedure. Guidelines for organizing the process are available. (Sections I.D.2 and I.D.4.)

6. Current data collection tools make possible the collection of a wide variety of performance data, but the data are often not the most effective types and frequently cause human and machine inefficiencies. For example, computer accounting data can be efficiently used, but accounting systems are frequently so unwieldy that analysts refrain from using the data. Improved accounting systems and associated analysis software are a major national CPE need. (Sections I.D.3 and I.E.2.)

7. Similarly, hardware monitors are somewhat difficult to employ because probe points are located deep in computer circuitry rather than being immediately available at a standard connector. Improved hardware monitor interfaces constitute another major CPE need. (Section I.E.2 and I.E.3.)

8. A multitude of important variables (arising from either measurement or modeling) must be examined in evaluating performance. Representation schemes such as Kiviat Graphs or simple models often aid analysts. (Sections I.B.1 and I.C.5.)

9. Models of computer performance frequently are

developed and applied without reference to empirical data, and data collection is usually performed without using any model to aid in experimental design. Modeling and measurement should be combined in both research and application efforts. For example, workload characterization schemes are needed that are based on good models and collectable data. (Sections I.D.5, I.E.4, I.E.5, and I.E.6.)

10. The objective of a performance analysis (whether emphasizing measurement or modeling) should not be to achieve the highest possible component utilization; in fact, higher utilization often implies reduced computing effectiveness. The objective of such an effort should recognize both the costs of a computer installation and the needs of users for service in a stable environment. (Sections I.A.2, I.B.1, I.B.2, and I.C.4.)

11. Difficulties in setting objectives for computer performance analysis are aggravated by the inconsistencies in objectives that already exist in large organizations; the existence of these inconsistencies must be recognized since determining a computerized systems performance is dependent on the choice of objectives. (Sections I.C.2 and I.C.3.)

12. The most attractive directions indicated for R&D efforts in the CPE area were toward developing and validating the underlying theoretical base for CPE; determining appropriate CPE measures and criteria; developing, validating, and refining CPE performance models; and improving workload characterizations. (Section I.E.6.)

13. Most CPE benefits could be realized with current technology. If this is to occur, an essential step is the development of a definite CPE reference work which provides clear definitions of CPE terminology, reliable information on the capabilities and limitations of CPE tools and techniques, and useful guidelines on how to apply CPE capabilities in different situations. One objective of this Report is to provide at least a stopgap candidate for such a reference work. But much more is needed. Two of the top-priority Workshop recommendations were that a group be set up to develop authoritative definitions for CPE terms, and that a definitive CPE Handbook be developed and published. (Sections I.D.7 and I.E.2.)

KEYNOTE ADDRESS TO THE NBS/ACM WORKSHOP ON PERFORMANCE EVALUATION

S. Jeffery

National Bureau of Standards, Washington, D.C. 20234

Everyone here has done a great deal of work and planning for this meeting. I wish to thank the ACM who joined with NBS in presenting this Workshop. We are also grateful to the National Science Foundation's Office of Computing Activities for their support in the planning phases and are very pleased that Dr. John Pasta of the National Science Foundation could join us.

You as a group, the leading advocates and authorities in the field of performance evaluation, have been selected to participate in this Workshop to address specific problems and formulate recommendations for making performance measurement and evaluation a commonplace and productive practice.

The sessions on performance factors, complementary pursuits, state of the art, and theory will address the range of performance measurement technology development. In your discussions, each of you should have in mind how the results of this Workshop can give impetus to the development of management understanding of the spectrum of application of performance evaluation.

Much attention has recently been focused on techniques and practices of performance evaluation in the technical literature and at various conferences and symposia. Notwithstanding, the uninitiated user has almost come to believe that mere procurement of a monitor will improve his system's performance (whatever that is). The participants at this Workshop should have as one goal the preparation of a guide for computer installation management which will explore the capabilities, limitations, and effective use of performance evaluation devices and techniques.

Your energies should be devoted to defining and recommending what is required to improve the state of performance measurement and analysis through standardization, education and training, and further research and development. In particular, we are looking forward to your recommendations for actions on specific areas in performance evaluation.

Some frequently-asked questions include:

- What are meaningful measurements?
- When is the appropriate time to measure?
- What are the criteria for the selection of these tools?
- Are there problem-specific tools?
- Can performance measurement lead to improved reliability or to a productivity index for management?

You are giving your valuable time here; I'm sure your objectives include the development of definitive statements answering specific technical and managerial questions about performance measurement. In the many instances where there are as yet no solutions to specific problems, the technical approaches to finding the solutions can provide the impetus for the necessary work.

I would like to bring to the attention of this group that with all the rhetoric about performance evaluation, there are actually fewer than 200 hardware monitors. These have been used by some estimated 1000 installations. A good many of these same installations are also users of the estimated 1500 software monitors being used. Thus, it would still seem that less than 10 percent of our medium and large scale system installations are attempting to apply software or hardware monitors to improve performance.

Perhaps it is a lack of knowledge about the proper use and role of performance measurement, and for that matter, about the devices and techniques, which has made management reluctant to apply these aids to better utilization. I have little information on the effective use of accounting data in performance measurement and would suggest this as an important task to be addressed.

I know this Workshop will provide the direction, framework, and thrust needed to achieve the cost effective utilization of our significant computer resources.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

CHAPTER I

Workshop Summary

Barry W. Boehm, Thomas E. Bell

TRW Systems Group, Redondo Beach, Calif. 90278

A. INTRODUCTION: WHY BE CONCERNED ABOUT PERFORMANCE ANALYSIS?

1. Potential Benefits

Computer Performance Evaluation (CPE) was selected as a subject of an ACM/NBS Workshop because of the significant leverage CPE activities can have on computer usage. Under the right conditions, CPE can be a major force for improving the efficiency and effectiveness of computer and related human operations. For example, the recent report by the General Accounting Office [1] [1] provides some good examples of the productivity improvements possible via CPE techniques from its study of their use at NASA's Goddard Spaceflight Center:

—The number of jobs processed by one computer was increased by 50 percent without increasing the number of hours the computer was used.

—The number of jobs processed by another computer was increased by 25 percent with a 10-percent increase in hours of usage and a 7-percent increase in the utilization of the central processing unit.

—Computer time worth $433,000 annually was saved through the use of these techniques and the acquisition of a more efficient compiler (a program that translates language used by the programmers into machine language). The one-time cost of making the changes was estimated at $60,000.

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

This report, and others on the subject [2–5] indicate that CPE techniques can lead to similar productivity gains, often estimated at 15–30 percent, on virtually all large general-purpose computers, most medium ones, and many small-scale ones. How appreciable a potential benefit this is can be seen by relating it to the recent AFIPS estimates [6] of a $26.5 billion installed base of general-purpose computer systems in the United States, growing to $33–48 billion by 1976. Currently, 74 percent of this value is in large and very large systems, 22 percent in medium systems and 4 percent in small systems. Using an amortization period of 4 years yields an annual expenditure of $6–7 billion on general-purpose computer hardware, and thus a potential national savings of over $1 billion per year.

2. Potential Difficulties

Leverage works both ways. There are also significant opportunities to misuse CPE techniques. Several Workshop participants were familiar with situations in which the only result of a $100,000-range CPE effort was a stack of unanalyzable data tapes. Other large efforts have foundered on impossible-to-validate model results. Others have been successful in achieving hardware efficiencies, but have thereby introduced more damaging inefficiencies in the organization's overall productivity because of increased job turnaround times, frequent changes in procedures and user interfaces, reduced flexibility to cope with emergencies, and the like. One participant indicated that he had initially tuned his computer system to a Central

Processing Unit (CPU) utilization of 90 percent, but found that users were extremely dissatisfied with the system's lack of responsiveness and its effect on their work. When he readjusted it to a CPU utilization of 65 percent, charging higher rates but providing more helpful and responsive service, users were almost universally satisfied.

3. Sources of Leverage

Potential savings and difficulties such as those cited above are possible primarily because of the way current (third-generation) computer systems are organized. These systems consist of a number of data processing, transmission, and storage access units which in theory can all be doing useful work at the same time.[2] In general practice, however, many valuable resources remain idle because of imperfectly organized resource management algorithms in the computer's operating system, imperfectly organized applications programs, incompatibilities between concurrently running programs, imperfectly organized human computer operator activities, and the like. By using CPE tools and methods, one can quite often pinpoint the resulting bottlenecks and formulate improvements which achieve significant increases in computer hardware efficiency. Often, though, the complexity of the computer system's interactions and the sensitivity to system changes of such services characteristics as response time lead to difficulties and disbenefits instead.

4. Objectives of the Workshop Report

These complexities and sensitivities are the reason that many of today's CPE activities encounter difficulties instead of payoffs. CPE practitioners, computer center managers, users, plant managers, auditors, government agencies, and others concerned must try to sort out a bewildering flurry of conflicting claims about CPE tools, conflicting definitions of CPE terms, and conflicting statements about CPE objectives, in the process of formulating their individual or mutual CPE plans. In the future, such CPE problems could become even more critical as computer architectures become more sophisticated, CPE products proliferate, and organizations become ever more dependent on reliably efficient computer performance.

There is thus a strong need for a readily available, definitive CPE reference work which provides clear definitions of CPE terminology, reliable information on the capabilities and limitations of CPE tools and techniques, and useful guidelines on how to apply CPE capabilities in different situations. One objective of this report is to provide at least a stopgap candidate for such a reference work. The Workshop participants recognize that this is the best that can be expected from their one-shot, volunteer effort; two of the top-priority Workshop recommendations were that a group be set up to develop authoritative definitions for CPE terms, and that a definitive CPE Handbook be developed and published—both of which would render most of this report obsolete.

5. Scope of the Workshop Report

The Workshop Report basically follows the structure of the Workshop sessions, which were organized around the following topics:

I. Performance Factors
   A. As Seen by User Groups: Commercial (Banking), Airlines, Government, Industry (Automotive), and Universities.
   B. As Seen by Level of Responsibility: Computer Subsystem, Computer System, End-User Subsystem Performance, End-User System Performance.

II. Complementary Pursuits: Validation, Reliability, Security and Privacy, Computer Design.

III. State of Art: Application Domains, Experimental Design/Data Analysis, Measurement Tools, Performance Variability.

IV. Theory.

V. Recommendations: Standards, Education and Training.

The "Workshop Summary" chapter continues with Section I.B discussing definitions and critical distinctions in the CPE area. Section I.C follows with a discussion of objectives and criteria by which to evaluate computer-based systems, followed by a summary evaluation of CPE tools and techniques in Section I.D. The summary chapter then concludes with Section I.E., stating the key issues identified during the Workshop and the resulting action recommendations.

[2] In earlier first-generation computer systems, only one component could be active at any one time. In second-generation computers, only a very limited amount of parallel resource usage was possible. These trends are elaborated by Warner in Section III.A.1.

The subsequent chapters introduce and present the papers contributed by the Workshop participants in the areas of objectives and criteria (Chapter II) and tools and techniques (Chapter III). Chapter IV elaborates on the key issues and recommendations, including the background papers and excerpts from the Workshop discussions of proposed recommendations.

6. A Final Introductory Comment

On encountering statements such as the above-mentioned "productivity gains of 15–30 percent" or "potential national savings of over $1 billion a year" due to CPE, the reader should heed this note of caution. The only savings that really count are the ones which appear on the bottom line of the balance sheet. Suppose, for example, that the "computer time worth $433,000 annually" saved by CPE techniques at NASA-Goddard were not used for some productive purpose. Then the Government has not saved any real money. In fact, it would be the loser by the $60,000 in real dollars spent on the CPE effort.

Thus, by themselves, CPE techniques serve only to make existing resources more available. Additional management actions are necessary to convert these into real dollar savings by deferring or cancelling planned computer acquisition or by returning or selling extra resources. Otherwise, unless the resources are used to run additional jobs which add comparable value to the organization, all of the impressive-looking CPE savings are illusory.

B DEFINITIONS AND CRITICAL DISTINCTIONS

1. Introduction

A good many well-meaning CPE activities founder because of a fundamental confusion between such terms as measurement and evaluation, capacity and activity, processor utilization and throughput, and others.[3] This section attempts to make those distinctions as clearly as it can, and to point out some of the more common pitfalls which may accompany improper distinctions. Much discussion of the Workshop was devoted to these distinctions and views often converged only after extensive discussion. Figure 1 presents a framework which tries to capture the essence of these discussions by employing a framework developed subsequent to the Workshop.

The major distinctions to be observed in Figure 1 are:

- The distinction between properties of applications systems (priorities, throughput, information value) and properties of computer systems (memory costs, channel capacities, CPU utilization).
- The distinction between the domain of values (profit, good will, manpower opportunity costs, evaluation) and the domain of activities (throughput, channel capacities, CPU utilization, measurement).

2. Components and Connections

Why do organizations develop and use computer systems? The process begins with a hypothesis that, for some application system(s), the relative value of computer-processed information to the applications system favorably compares with the costs of computer processing. This leads to a commitment of expenditures (in dollars and in opportunity costs for scarce personnel, floor space, etc.) to procure a computer system capable of processing the necessary information.[4]

In fact, however, several separate items are acquired:

- A potential hardware capacity for each device and other resource in the computer system (e.g., a CPU rate of a million additions per second).
- A hardware architecture which yields an effective hardware capacity which may not be the sum of the individual capacities in the system (e.g., channel activity may degrade the CPU rate by 5 percent).
- An availability assurance activity which, via scheduled maintenance, hardware and software reliability and recoverability efforts, attempts to make computer resource availability as high and as predictable as possible (e.g., scheduled hard-

[3] Also, a good deal of time was consumed at the Workshop, as elsewhere, in determining or discovering how other people defined various key terms.

[4] For simplicity, the activity (and related costs) of application software development is considered as another application system activity— which it is, of course, when one considers its interaction with computer system performance in such areas as debugging run response time.

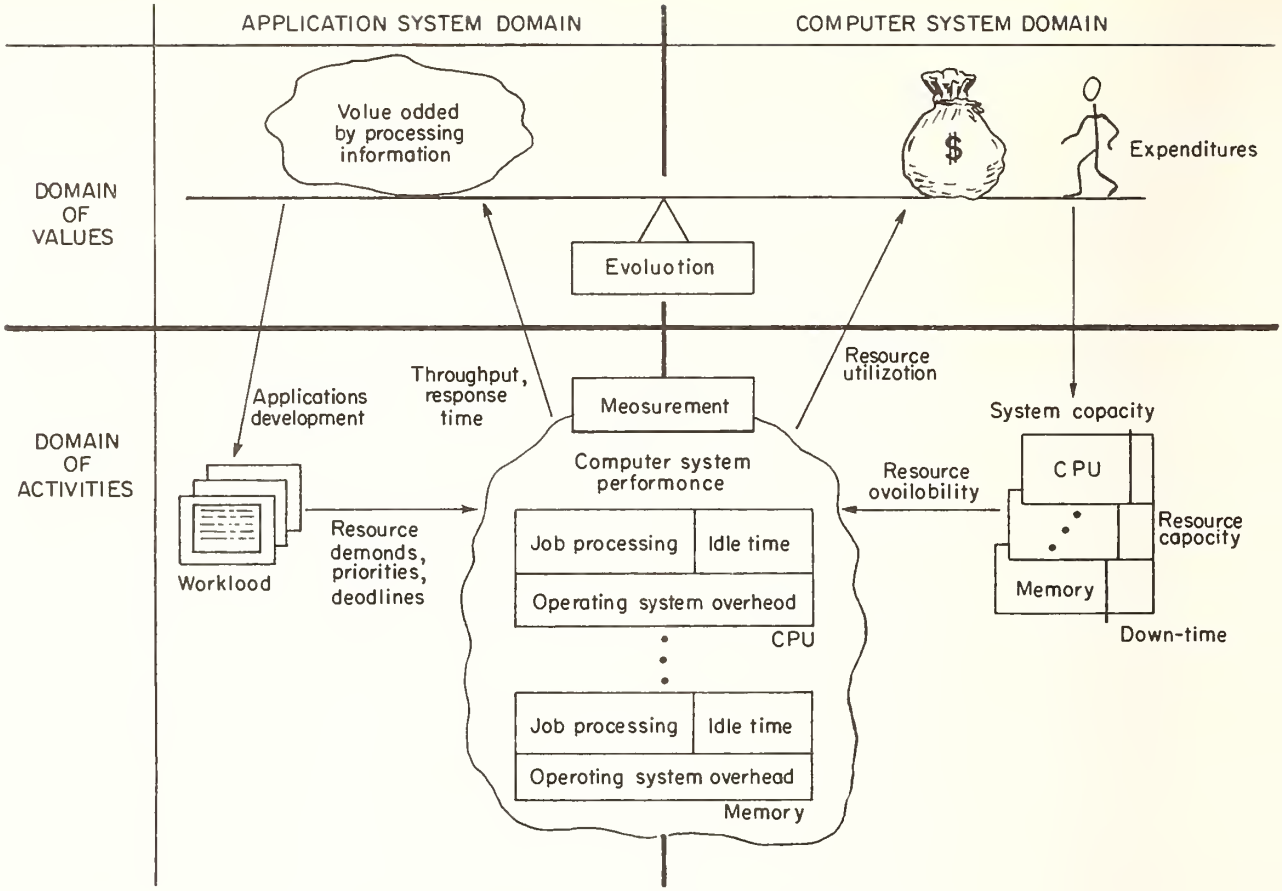


Figure 1. Computer performance analysis domains

ware maintenance preempts other computer activities between 7 and 8 a.m.).

- An operating system which works with the computer resources available and the sequence of resource demands, priorities, and deadlines embodied in the applications workload to produce a throughput of processed jobs or transactions, each with its own response time from submission to completion. The operating system consumes some (often difficult-to-determine) portion of the available resources as overhead, some portion is devoted to applications job processing, and the remainder is idle time for resource.

At the Workshop, Hughes advanced a scheme for tying some of these ideas together from the viewpoint of the computer. He suggested that an analyst should consider a multiprogrammed computer with adequately large core to sustain a complement of peripheral devices capable of a large (but finite) number of transfers per second. This situation is represented in Figures 2a through 2g.

Figure 2a shows a two-dimensional space in which we may plot executed instructions per sec and I/0 transfers per second. Clearly, as shown in Figure 2b, this space is bounded by some potential resource capacity (Cm,Tm) for the specific configuration. As we use the system and execute programs there will be some mean free path, say m instructions, at which interval an I/0 transfer will be initiated. This is modelled (Figure 2c) by the user load line OP such that the slope $m = \frac{\text{Actual Instructions/sec}}{\text{Actual Transfer/sec}} = \frac{Ca}{Ta}$. Furthermore, Tm–Ta = potentially available but unused I/0 transfers; Cm–Ca = CPU instructions available but not utilized by the user program(s).

This difference Cm–Ca may be further sub-divided as in Figure 2d. Cm–Cs represents a CPU operating system standing overhead; Cs–Ci represents a CPU

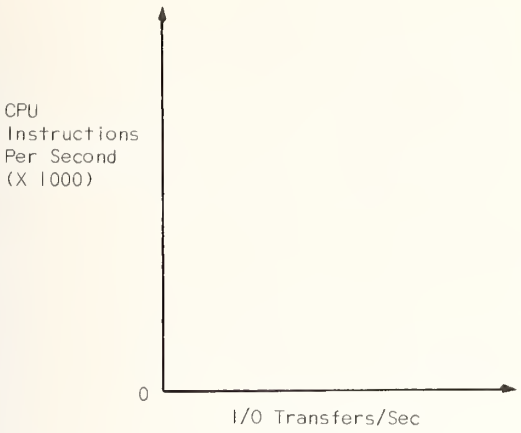


Figure 2a.

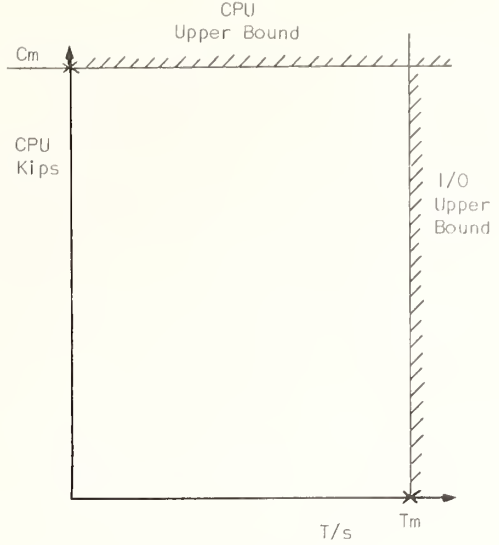


Figure 2b.

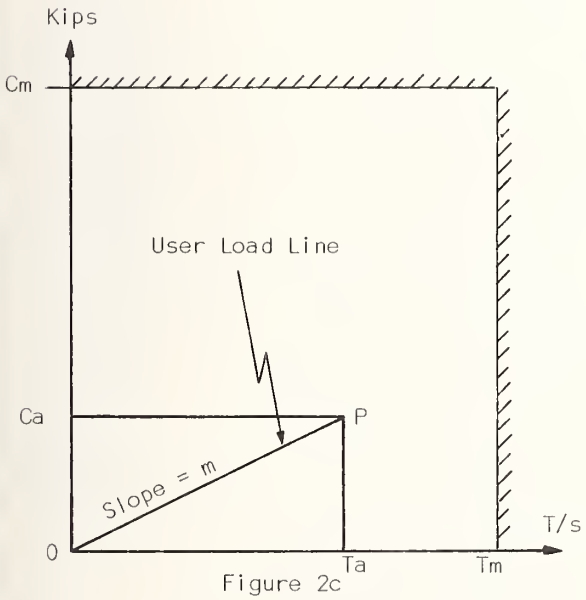


Figure 2c

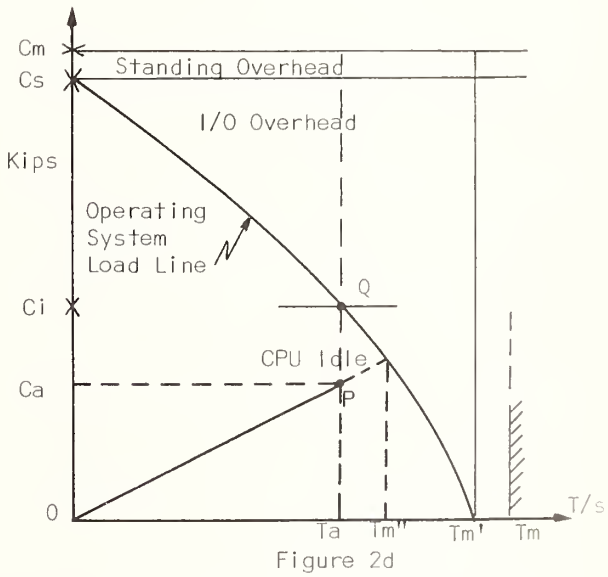


Figure 2d

operating system dynamic overhead which is some function of (multiple) user program activity. In general, it will be proportional to I/O activity but will in practice include the performance of other services initiated by users. Observation of such a system shows that the line CsQTm′, though linear for low values of T, sags markedly as T increases. This is due to the increased burden of longer table and queue searches, more flags to set and unset, more file opening/closing, more I/O contention for core memory etc.

Thus the operating system-imposed upper bound on I/O (Tm′) may well be less than the theoretical Tm, and for a given user program, the upper bound will be Tm″ (less than Tm′).

Direct measurement of workloads will also show that real user load lines tend to fall into two distinct classes: scientific and commercial. Figure 2e shows this: FORTRAN executions and assembly language coded scientific problems generally have a slope which is a factor of five to ten greater than that of commercial programs, which include not only COBOL executions but also compilations and assemblies.

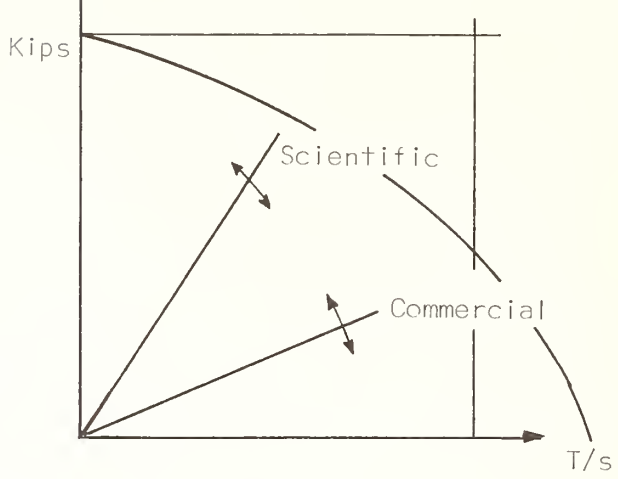
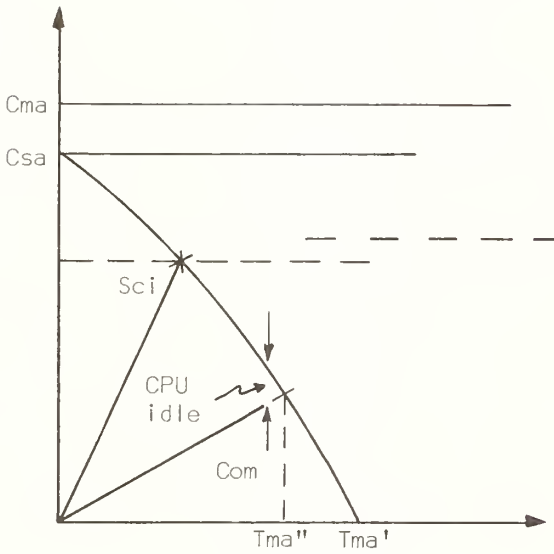
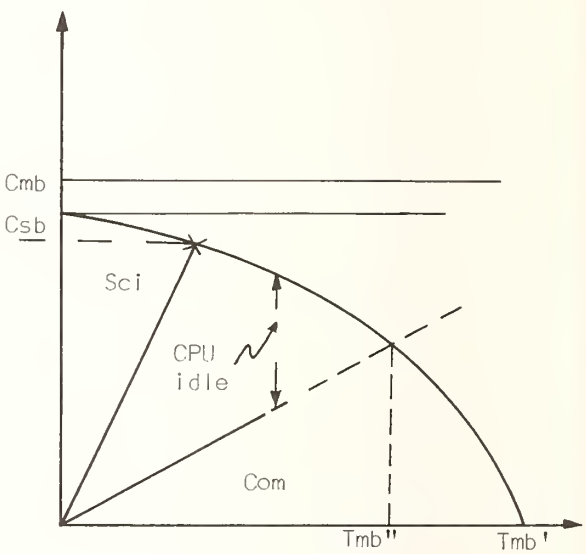


Figure 2e



Computer A

Figure 2f.



Computer B

Figure 2g.

Having established this simple model, now consider two computer systems A (Figure 2f) and B (Figure 2g), and for argument's sake suppose them to have the same user instruction repertoire and peripheral configuration but different sets of privileged instructions, different I/O and priority interrupt structures and different system programs and operating systems.

As shown, Computer A is approaching saturation for commercial programs with little reserve of CPU idle time; for scientific programs, it has hit a ceiling (the monitor load line) and has no idle time. Despite having (say 20 percent) less raw CPU power, Computer B can process commercial work as fast as A, has more idle time, less operating system overhead, and can process scientific programs say 10 percent faster before saturating. Also, on reconfiguration, Computer B could support a practical I/O transfer load Tmb″ some 30 percent greater than Tma″.

This graphical model warns us not to look in isolation at one or another aspect of a computer system's performance; the tortoise may be outperforming the hare.

3. Performance Improvement Avenues

These structures can be employed in a variety of practical problems, including the most popular one of improving the performance of an existing system. It is evident that there are a number of avenues toward computer system performance improvement, in terms of achieving a more favorable balance of value added by processing information to computer system expenditures. These avenues are enumerated below.

1. Within the "computer system performance" area of Figure 1, one can improve the operating system's scheduling rules so that concurrent jobs compete less often for the same resources, one can ensure that the most frequently used programs are in the most rapid-access storage device, one can reduce operating system overhead. To make such improvements effectively requires information on the performance characteristics of the applications and the components of the computer system; measurement can involve obtaining such information via electronic devices called hardware monitors or via special computer programs called software monitors.
2. Within the computer system domain, one can monitor resource utilization and replace underutilized resources with cheaper ones. One can increase resource availability through better computer operator procedures, reliability and maintainability provisions, etc. One can also provide more system capacity via architectural changes. These fall in the category of computer center actions.
3. Within the applications domain, one can reorganize application programs to achieve the same objectives via more efficient sequences of resource demands. This process, here termed application program improvement, is often aided by hardware monitors or special software monitors. Other improvements in classes of applications programs can be obtained via standards and procedures (e.g., for program and data organization and storage) promoting more efficient or compatible resource sharing; these are termed direct-user provisions. Finally, one may provide additional opportunities to end users to balance the cost and value of their processed information via such economic measures as direct charging for computing time or computer-generated management reports, variable pricing for quick turnaround, idle time, etc; these are called end-user provisions.

One of the activities at the Workshop was to have participants estimate the savings achievable by these and such other CPE activities as modeling, self-monitoring, and standards, both in terms of long-term potential and in terms of savings achievable within the next 3 years, measured in terms of the percentage of current hardware expenditures which might be saved.[5] (Overall, in the United States, current hardware expenditures total about $6–7 billion per year.) These estimates, summarized in Table 1, indicate primarily that user and application program provisions have the greatest improvement potential, but also show that most CPE activities exceed the pace of computer-hardware technology as a source of future savings.

TABLE 1.—*Results of workshop poll on relative potential of various CPE activities.*
(Medians of participants' very rough estimates of savings realizable, as a percentage of current computer-hardware expenditure)

Realizable Within Next 3 Years	Long-Term Potential
VERY LARGE (30%*)	VERY LARGE (40–60%*)
Applications Programs	Applications Programs
LARGE (15–20%)	Direct User Provisions
Software Measurement	LARGE (30–40%)
MEDIUM (10–15%)	Software Measurement
Self-Monitoring	End User Provisions
Direct User Provisions	MEDIUM (20–30%)
Hardware Measurement	Hardware Measurement
Computer Center Activities	Computer Center Activities
Pace of Technology	Self-Monitoring
NOMINAL (5–10%)	NOMINAL (10–20%)
End User Provisions	Pace of Technology
Modeling	Modeling
Standards	Standards

*The estimated savings might come from sources in addition to the computer-hardware budget. The figures are not additive because each category's saving was estimated under the assumption that no other category of CPE activity was performed.

4. Operational Significance of Distinctions

If the above categories and distinctions are carefully kept in mind, a number of troublesome conceptual

[5] The estimated savings might come from other sources than just the computer-hardware budget. Each activity was considered independently in estimating savings; thus, the savings are not additive. Such projections were, of course, merely guesses; however, they give some indication of how participants ranked items in importance.

pitfalls and their corresponding operational penalties can be avoided. These include:

1. Incomplete consideration of improvement alternatives. One Air Force business data processing situation was presented as an example of the need for considering a variety of alternatives. The effect of monitoring activities was positive and appreciable, but for some activities, greater gains were made via the direct user provision of relaxing core residence limits from their previous 20K bytes. When this was done, one common job which previously ran in about 220 minutes elapsed time with 3057 overlay loads of 19K bytes each was reorganized to run in about 11 minutes elapsed time with overlay loads of 21.5K bytes each.
2. Overcentralizing cost-benefit evaluation decisions. It is difficult for top-level computer system managers and application system managers to establish priorities, standards, and procedures which can closely track the time-varying utilities of many users. Often, for example, a user would be more than willing to pay for priorities or special arrangements if only the rulebook would let him. The more the cost-benefit evaluation function can be distributed among individual users (within compatibility guidelines) via pricing and other economic provisions, the more the individual user can feel responsible, accountable, and motivated to improve his applications' overall performance.
3. Confusion between capability to process data (capacity) and amount of processing (by resource: utilization; by job: throughput). One common manifestation is to assume, for example, that buying twice as much processing capacity will produce twice as much throughput. As the additional capacities are connected via the the complexities of operating systems and workload sequencing, this assumption always turns out to be invalid. So, generally, are other assumptions resulting from considering capacity and throughput to be indistinguishable.

5. A Critical Distinction

There is one distinction which is more important to make than all the others. Far too many performance improvement efforts have failed to appreciate the distinction, and have therefore led themselves into well-intentioned but highly unsatisfactory outcomes. This critical distinction is that:

Determining component utilization is measurement, not evaluation. Evaluation takes place with respect to the objectives and goals of the organization and usually includes some measurement. A high CPU utilization in itself is not necessarily the best thing for an organization. In many situations, organizations have achieved higher CPU utilization and improved the organization. However, in many others, organizations have sought and achieved higher CPU utilizations, but in the process have degraded turn-around time, peak-load capacity, staff morale, or other factors which contributed more to the organization's goals than high utilization of the CPU—or of any other computer center resource.

C. OBJECTIVES AND CRITERIA

1. Introduction

The difficulty of relating computer system activity measurements to applications system performance (and therefore the computer's utility) can perhaps best be illustrated by citing an extreme case. In this example, computer-system efficiency, reliability and throughput were near zero, but the system still ranks as one of the most successful computer applications to date.

Several years ago, the income tax agency of a large developing nation acquired a huge computer system to process tax returns. Unfortunately, since software, personnel, and other operational considerations had not been thoroughly planned for in advance, the system was hardly ever available to process the first year's tax returns, and its performance on those was extremely inefficient.

However, the astute tax officials proceeded to feature the new tax computer center in a barrage of television, radio, newspaper and magazine features, which portrayed the computer as ready to catch anyone who understated his income or tax liability. The results in the applications arena? An increase in tax revenues of over 400 percent.

In this situation, virtually nothing that could be obtained with a hardware or software monitor would

contribute significantly to the evaluation of the computer system's effectiveness in producing an excess of revenue over expenditures. Again, however, this is an extreme case, and even here the situation has evolved to a point that revenues have stabilized, and efficiency in processing returns has become a key determinant of net income, in which case computer resource utilization improvement and associated monitoring activities become quite important.

2. Consumer Concerns

The above example is particularly valuable in pointing out that computer hardware cost reduction is not necessarily the sole or primary objective of CPE efforts. A related consideration is that a great deal of attention must be paid to the effect on other components of computer system performance when performing a computer system tuning activity. To get an idea of the nature and relative importance of these other performance components, workshop participants were asked to complete the following statement:

> I'm a consumer of computing services. I don't mind if you tune the system, as long as you don't degrade my _____.

The summary results from the 17 respondents are given in Table 2.

TABLE 2.—*Results of consumer concerns poll*

"I'm a consumer of computing services. I don't mind if you tune the system, as long as you don't degrade my _____."

Consumer interest	Number of Responses
Personal job performance (productivity, ability to do job)	7
Ease, ability to use system (test, debug aids, relearning, ability to run old jobs)	7
Cost for my type of service (even though average cost decreased)	7
Response (turnaround) time	6
Profit	3
Reliability	3
Service	1
Service variability	1
Major customer's job completion	1
Throughput	1
Cost variability	1
File security	1
Priority	1
Empire	1

Besides these concerns, Workshop participants identified other important considerations not so closely linked with system tuning. Four of these considerations are discussed by papers in Section II–C. The paper by Wilner points out some of the major CPE considerations during system design, particularly in detremining hardware / firmware / software tradeoffs. The paper by Rubey indicates some of the tradeoffs between system performance and program validability, often a more overriding concern, particularly in "man-rated" computer systems such as are used to support the manned space flight program. The paper by Hughes furnishes more detail on the complementarities between performance monitoring and reliability activities. The paper by Chastain discusses the considerations relevant in balancing system performance and monitoring ability with data security assurance—concluding, for example, that it is not likely that performance monitors would unintentionally compromise sensitive information from secure computer systems, but that it is possible in many situations for performance monitors to intentionally obtain sensitive information.

In this regard, the computer's relation to the organizational goals is no different than that of the telephone, elevator, warehouse, or other facilities operated by the organization: its efficient utilization is not an end in itself, but one of a number of means toward improving performance with respect to overall organizational objectives and criteria. These, and their relation to computer systems, are discussed in the next section.

3. Relating Measurement to Evaluation: General Considerations

Perhaps the most important thing that could be done in the CPE area would be to determine a set of functional forms for relating computer measurements to an evaluation of the computer system's contribution to its application systems. However, this is extremely difficult in general, because measurements are usually made on the performance of computer subsystems (CPU, memory, channels, etc.) and these are related in complex ways to the performance characteristics of the computer system as a whole. These in turn relate in complex ways to end-user subsystems such as point-of-sale terminal operations, which in turn are related in complex ways to end-user system criteria such as profit, good will, etc. Performance analysis

considerations at these four levels are discussed in detail in Section II–B in the papers by Carlson (computer subsystem), Morrison (computer system), Kiviat (end-user subsystem), and Nielsen (end-user system).

A particular difficulty is that often what looks like a performance improvement at one level actually degrades performance at higher levels. For example, in Kiviat's point-of-sale terminal example, one can create a situation in which better end-user subsystem performance (zero bad debt rate via on-line credit checking) can correspond to poorer end-user system performance if the credit checking delays cause customers to stop shopping at the store. Similarly, customer delays at the end-user subsystem level could be caused by overconcentrating on performance at the computer system level. One typical way would be to increase computer system efficiency by requiring clerks to batch their inputs in a way that simplified computer transaction processing but made it much harder for the clerk to correct data entry errors. And, as Carlson points out, running programs out of low-speed core can produce very high values for CPU utilization (computer subsystem performance) while degrading computer system performance.

4. Relating Measurement to Evaluation: Some Specifics

However, there are some situations in which a fairly straightforward functional form relates computer performance to application system performance. By far the most common is the situation in which most of the workload consists of periodic equal-priority status reports or numerical calculations for which there is very little time-criticality for the results. In this situation, computer system performance and applications system performance are identical: reports or jobs processed (throughput) per dollar spent on the computer system.

This situation can thus be characterized in terms of a relationship between the value of processed information and the time it takes to complete processing it. At least three other common situations may be characterized by a relationship of this form. In each case, the value-versus-time curve provides a key for relating computer system performance to application system performance. Figure 3 illustrates the four common types of value-versus-time relationships, which we describe in more detail below:

Type A: Accumulated Batch: Low Priority.
Slight, uniform decrease in information value. This is the situation discussed above. Scheduling decisions are made only with respect to load-balancing for throughput.

Type B. Batch: High Priority.
Larger, but still uniform (linear) decrease in information value. This decay curve is typical of software development situations, where the time lag is turnaround time on debugging runs. Here, the appropriate criterion is some mixture of throughput and turnaround time per dollar. Scheduling decisions consider throughput, but priority is also given to short jobs over long jobs.

Type C: Conversational.
Non-linear decrease in information value. The decay curve shown is typical of some interactive systems, where human-factors data (e.g., Reference 7) indicate falloffs in sustained concentration when delay times begin to exceed 2–3 seconds and, at another level, 15–25 seconds. Here an appropriate criterion is "responsiveness" per dollar with respect to the decay curve; an example of a scheduling system built to accommodate it is given in Reference 8.

Type D: Deadline-driven.
Step-function decrease in information value. This is typical of various "real-time" applications such as spacecraft mission control, industrial process control, or banking situations in which missed deadlines can mean large losses of float, or interest dollars on the transactions processed. Here other criteria inconsistent with throughput per dollar such as extreme reliability and reserve capacity to meet peak load situations become important; References 9 and 10 give thorough treatments of the additional considerations here.

Of course, no installation is a pure example of any of the curves in Figure 3. Installations with many routine file updates or low-priority scientific data reduction jobs generally have a Type A workload but not completely. Many others, including some of the

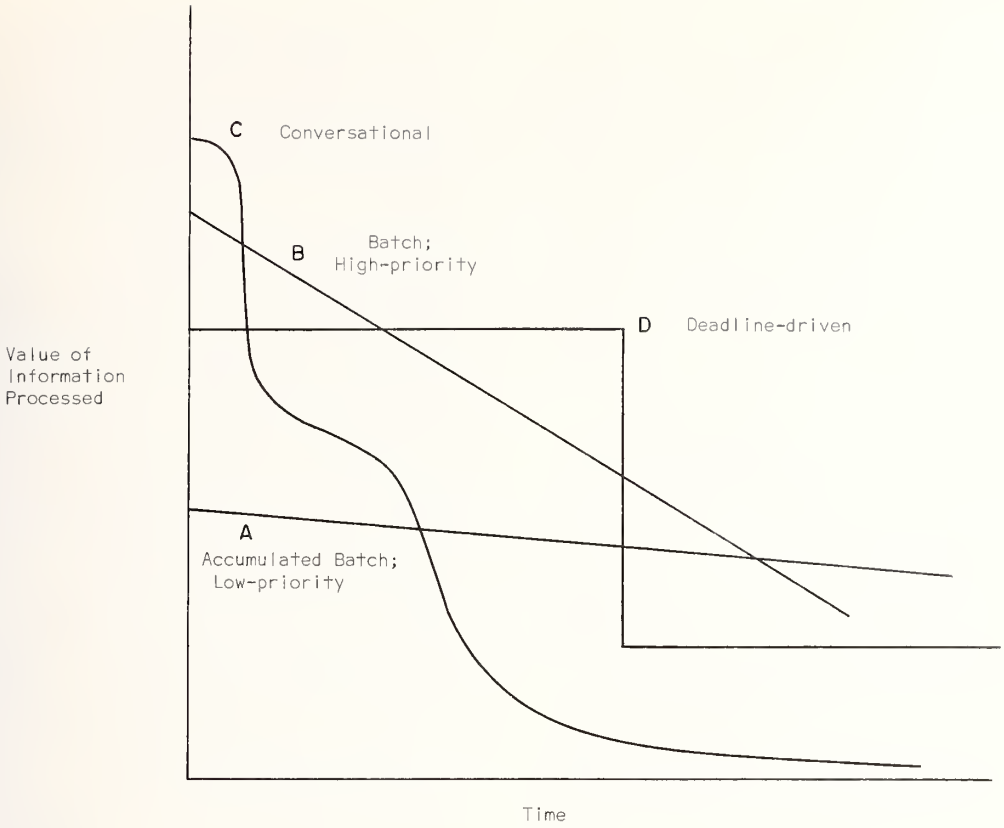


Figure 3. Some Characteristic Relations Between Computer Processing Time and Information Value

General Motors systems discussed by Seals in Section II.A.2 of the report, approximate Type B during the day and Type A at night. Others discussed in Section II.A of the report include Browne's discussion of University computing, often a mix of Types A, B, and C, and DiNardo's discussion of bank data processing, which can be a mix of Types A, B, C, and D.

Also, any CPE effort must include not only time-oriented criteria but also the additional consumer considerations enumerated at the beginning of this Section. Thus, at this point the types above provide only a conceptual aid to CPE efforts, but they could become more. One of the desirable efforts discussed at the Workshop was a project to achieve more definitive characterizations of the value aspects of common types of workload, including not only time dependencies but also other typical functional relationships.

One particularly important relationship to illuminate is the tradeoff between hardware and software development and maintenance efficiency since, for many organizations, software costs are two to three times higher than hardware costs. Figure 4 [11, 12] shows the results of a data-collection effort for airborne computer software, indicating that extremely high hardware utilization figures correlate with highly escalating software costs. Similar experiences with ground-based installations in government and industry indicate that this type of curve also characterizes other software development and maintenance activities. However, particularly when a computer system is sized and tuned for production, it is often done as if the "folklore" curve were true, leaving hardly any excess capacity for software development aids, test packages, or orderly expansion of applications software. Thus the gains in hardware savings can be more than eaten up by escalating software costs.

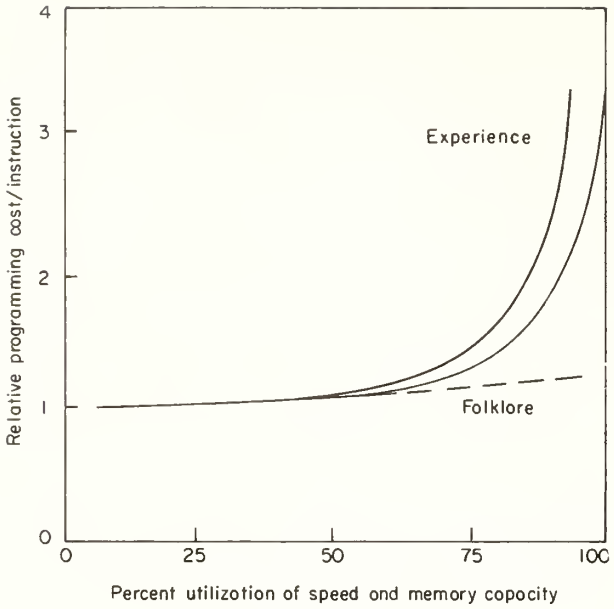
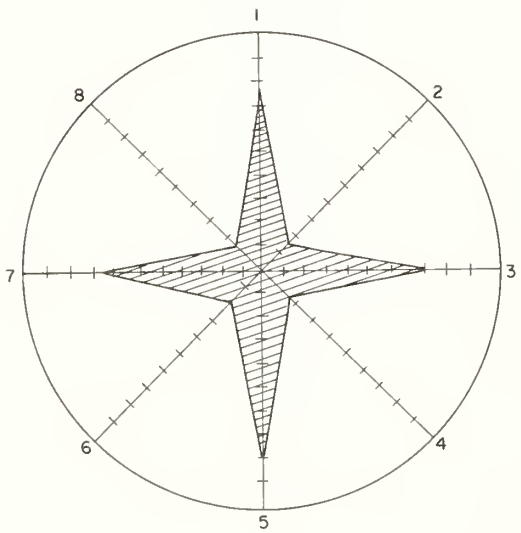


Figure 4. How do hardware costs affect software?

5. Multivariate Performance Criteria

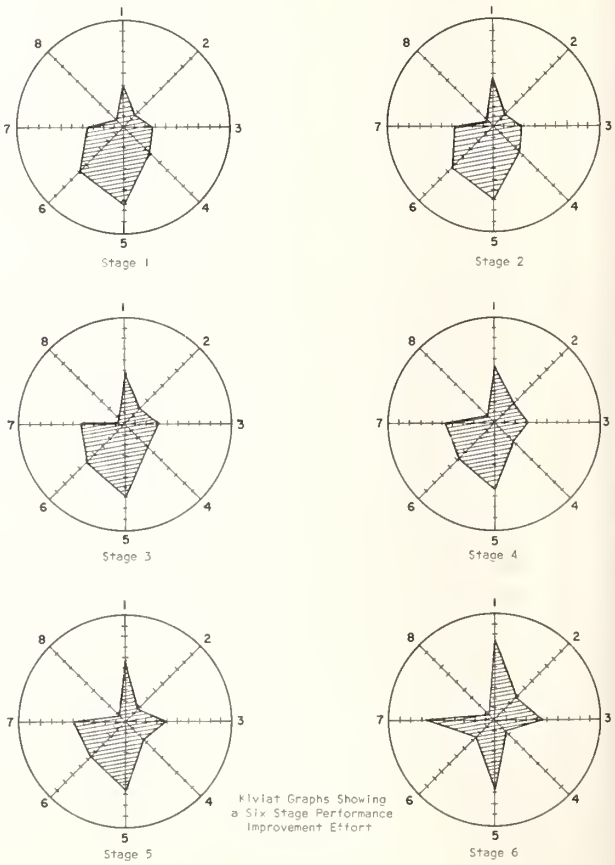
Another difficult problem discussed at the Workshop was that of balancing performance in a number of dimensions. Even in banking, dollars are not a "universal solvent" with which the value of all the other criteria can be expressed.

Various schemes were advanced for making multivariate performance data easier to assimilate by decisionmakers, including various types of bar charts, and shaded or multicolored score cards. The most attractive and novel scheme was one advanced by Kiviat, in which a number of variables are each displayed radially. Kiviat indicated that he had seen the technique used in the medical field to display multiparameter information about patients (see for example Reference 13). Since the Workshop, a great deal of activity [14, 15, 16] has been devoted to determining appropriate sets of axes for such charts, now called "Kiviat graphs". Figure 5a shows an example with a particularly good choice of 8 axes, in which the desired system performance takes the shape of a star.



	Activity	Units		Activity	Units
1	CPU Active		2	CPU Only	
3	CPU/Channel O'lp		4	Channel Only	
5	Any Channel Busy		6	CPU Wait	
7	Problem State		8	Sup'vsor State	

Figure 5a. Kiviat Graph



Kiviat Graphs Showing a Six Stage Performance Improvement Effort

Figure 5b

Figure 5b shows the results of a six-stage CPE effort illustrated via Kiviat graphs; [17] the intuitive impact of the technique is striking, particularly when compared with the same data presented tabularly in Table 3.

TABLE 3.—*Tabular data from Kiviat graph*

Axis	Percentage Measure	Measuring Stage					
		1	2	3	4	5	6
1	CPU active	40	46	49	52	56	75
2	CPU only	13	17	18	22	17	28
3	CPU/channel overlap	27	28	32	31	40	47
4	Channel only	38	35	31	26	25	15
5	Any channel busy	74	70	69	61	65	62
6	CPU wait	60	54	51	48	44	25
7	Problem program state	35	37	41	47	49	66
8	Supervisor state	4	6	6	4	6	8

Often, though, some of the important performance considerations are, as Nielsen explains in detail in Section II.B.4.

> * * * difficult to measure or judge, or simply too fuzzy * * * it may be very difficult to obtain a user's indifference curve between lower probabilities of service disruption on the one hand and higher cost with greater customer satisfaction and greater staff efficiency on the other.

Given such difficulties, often the best that can be done is to make cost comparisons for some informally-defined level of performance. In such situations, one can often use measurement and analysis tools to reduce costs while keeping performance roughly equivalent. The next section discusses the types of CPE tools and techniques and their relative capabilities and limitations in performing this function.

D. TOOLS AND TECHNIQUES

1. Introduction

A large complement of tools and techniques is available to aid in CPE efforts. Each has its own range of potential insights into performance improvement, and its own resource requirements in terms of added computer overhead, required personnel time and expertise, lease or purchase cost, etc. One of the major CPE problems is that of deciding which sequence of activities will produce the most cost-effective set of insights into performance improvement.

Therefore, this section will begin by outlining some of the methodological tools available for charting one's course through a CPE effort, followed by summaries of the capabilities available for data collection (monitors, accounting systems), data analysis (statistics, data representation), modeling (analytic, simulation), and management (standards, economics). The section ends with a discussion of priorities for research and development of improved computer performance analysis tools.

2. Methodological Tools and Techniques

These techniques provide guidelines for organizing a CPE effort so that it is guided by objectives and criteria rather than by data considerations, and so that efforts are not begun which could have been predicted in advance not to provide much illumination. Often, though, the opposite is the case: [18]

> After an installation decides that it should be concerned with performance improvement, the most common step is to examine available software and hardware monitors. Salesmen present their products, and one is selected. It is procured and personnel are assigned to begin measuring the system with the unfamiliar tool. This is very expensive; monitors tend to be costly, and personnel must be diverted from other work to the new activity. To make matters worse, the measuring process usually severely disrupts machine operation. The payoff for the expense is expected to come from improved performances resulting from the implementation of a system modification. Unfortunately, this reward is seldom realized. Instead, the procedure resembles the flow-chart below:

In has almost become a rule of thumb if a CPE effort is begun with the question "Shall I get a hardware or a software monitor?", that the effort will fail. Below are sketched two methodological tools which provide more assurance of success in improving an existing system's performance. They are the "systems analysis" approach and the "scientific method" approach.

The "Systems Analysis" Approach

This approach provides a set of general guidelines for avoiding pitfalls in the analysis of complex systems. It involves a multi-step, iterative approach, emphasizing the explicit formulation of objectives, criteria, assumptions, and alternatives before proceeding with detailed data collection and analysis. Figure 6 illustrates the usual sequence of steps, generally beginning with "Formulating the Problem." The approach is described in more detail in Reference 19.

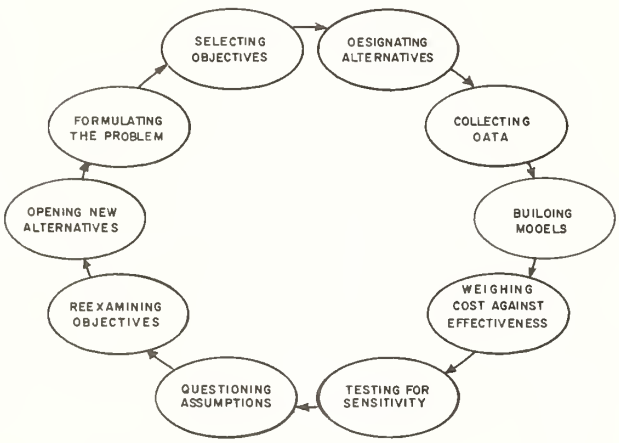


Figure 6. Activities in Analysis

An important point on these or other similar approaches is that they cannot be formulated in enough detail to show an inexperienced analyst precisely how to formulate incisive hypotheses, structure efficient data collection procedures, or assess the relative value of several performance variables with respect to the organization objectives. At the Workshop there was some discussion on the amount of practical experience that was necessary to properly run a CPE effort. The consensus was that the learning period for a recent B.A. or B.S. in computer science would be about two years. This topic will be discussed further in the Education and Training portion of Section V.

The "Scientific Method" Approach

This approach is based on the time-honored sequence of observation; hypothesis formulation; hypothesis-oriented experimentation, data collection and analysis; and iteration as embodied in the scientific method and illustrated in Figure 7. A short explanation of each phase follows; more details can be found in Reference 18. A similar approach is advocated in Reference 20.

Understand the System (Phase 1)

The first phase of a performance improvement effort involves understanding the particular computer system in terms of management organization of the installation, characteristics of the workloads processed by the computer system, descriptions of the hardware configurations and software programs in use, and information as to what computer-usage data are collected.

Analyze Operations (Phase 2)

The second phase involves the collection of more detailed data to analyze operations. These data, more quantitative than the data collected in the initial phase, provide an analyst with sufficient information to analyze and evaluate the performance at most computer installations. In addition to analyzing operations, data collected in this phase can be useful in reviewing the operational objectives of the installation. Such objectives may include rapid on-line response, low costs, flexibility, easy-to-use software, and good batch turnaround.

Formulate Performance Improvement Hypotheses (Phase 3)

Based on system inefficiencies and/or bottlenecks identified in the analysis of operations (phase 2), hypotheses about probable problems and possible cures can be formulated. These should be specific and performance oriented.

Analyze Probable Cost-Effectiveness of Improvement Modifications (Phase 4)

Before hastily gathering data to test a hypothesis, it is important to analyze whether the resulting performance improvement would be worth the investment. For example, consider the possible hypothesis:

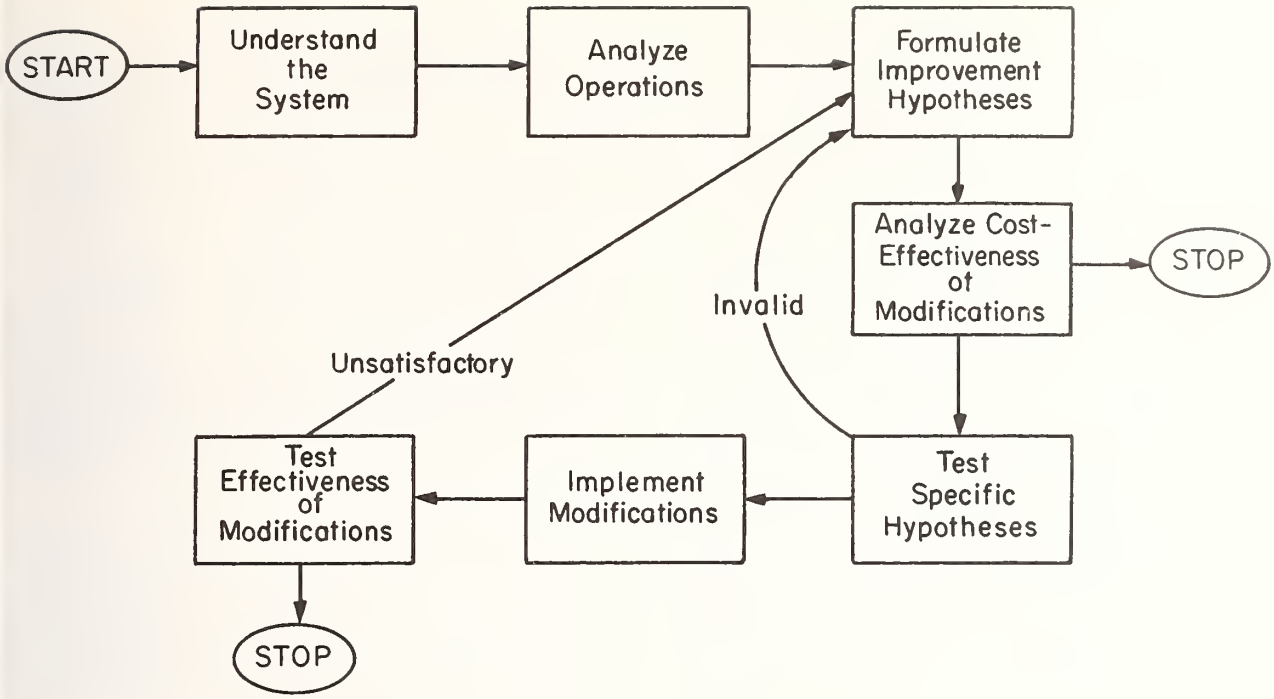


Figure 7. Suggested Performance Improvement Procedure

"If we add another $60,000 worth of communications equipment, we can probably reduce response time from 2 seconds to 0.1 second on our job-query terminals."

Since in most situations a 2-second response is acceptable to practically all terminal users, it would be difficult to justify the cost-effectiveness of such a modification.

Test Specific Hypotheses (Phase 5)

Although short studies (two or three days) usually devote little time to testing specific hypotheses, the bulk of an extensive performance improvement effort is devoted to this phase. The discussions later in this section on Data Collection and Data Analysis expand on these considerations.

Implement Appropriate Combinations of Modifications (Phase 6)

Several modifications are often simultaneously implemented because the effort required may be about the same as the effort for only one. Care must be taken that an installation can stand multiple changes without undue impact on production. Also, additional care must be exercised so that modifications do not cancel each other.

Test Effectiveness of Modifications (Phase 7)

Utilizing the measurement tools, data-collection techniques, and test designs used to test specific hypotheses (phase 5), the effects of modifications on performance must then be tested. Modifications may result in satisfactory improvements in performance, but often further modifications are necessary to achieve the desired effect. A recycling through the process (starting in phase 3, the formulation of performance improvement hypotheses) will be required.

3. Data Analysis Tools and Techniques

At one time simply collecting any performance data presented a legitimate problem. Early techniques included using devices that were little more than con-

ventional counters with some minor logic. Such monitors were not easy to use. Early software monitors were limited to collecting a few seconds or minutes of data because of the overwhelming costs of data reduction; one monitor required up to 7½ hours to reduce 5½ minutes of trace data to usable information.

Since the early devices, dramatic improvements have been made. The tools have been made easier to use, and some monitors now possess both hardware and software capabilities. These monitors have been used to collect data on a variety of systems, and some generalizations about results are now possible.

One of the difficulties confronting newcomers to the performance analysis field is a dearth of information on typical performance of the computer system components. A most useful contribution in this regard for IBM 360 series computers is presented in the paper by Gary Carlson in Section II.B. Table 4 shows the basic results of his study.

The latest hardware monitors often use minicomputers to reduce data during collection. On the other end of the price spectrum is a monitor employing a CRT to display dynamically the utilization of 8 or 16 logical resources. The future will probably see monitors integrated into, or communicating intimately with, the host computer. Warner in his paper in Section III.A, suggests that continuous data interpretation will become a reality [6] rather than the current occasional measurements.

Accounting systems, of course, can provide continuous data on performance. Since workload characteristics are critically important in performance analysis, the resultant data should be extensively used for workload analysis in addition to determining resource utilization. However, several problems limit the data's usefulness.

First, accounting systems are designed for accountancy. Therefore, they aggregate data in ways that aid billing but impair performance analysis. Often, for example, system overhead charges are spread in arbitrary ways across all jobs without indicating the resources used by each job itself.

[6] Subsequent papers have shown this to have become a reality already, e.g. Reference 16.

TABLE 4.—*Typical range of hardware monitor measurements as % of wall clock time*
Gary Carlson, Brigham Young University

	360/30	360/40	360/44	360/50	360/65
System Meter	21–80	56–100	83–88	93–100	90–98
CPU Busy	16–51	17–52	32–48	44–85	32–93
CPU Only Busy	16–34	15–43		21–42	
Protect Key 0	19–24	18–27		38–62	
Supervisor State	20–32	22–67		30–53	25–32
Problem State		20–47	2.5–4.5	33–40	
CPU Wait On Printer	8–10	6–17	5–26		
CPU Wait On Disk Seek	3–9	2–52	3–11	1.1–13.8	
Multiplexor Busy	0.8–2.1	0.5–1.2	1.0	0.8–1.4	1.9–3.1
Multiplexor Meter In		10–26	61	40–100	
Selector Channel (tape & disk)	16–27		21–35	31–49	13–44
Selector Channel (tape)	3–15	2–23		9–13	
Selector Channel (disk)	5–19	11–18		25–39	11–48
Selector Channel (teleprocessing)					8–57
Printer Busy	8–22	7–36	18–27	29–53	
Reader Busy	3–16	5–16		11–21	
Cards per Minute	919–1033				
Lines per Minute Rated 1100	813–1077	936–1074	959	975–1140	
Lines per Minute Rated 600				570–683	
Emulate Mode	0–6		21–34		6–36
LCS Inhibit				16–30	

Second, accounting systems appear to be rather low on the list of items that vendors feel are important. A number of idiosyncrasies and plain errors are common in most systems.

Third, accounting systems use inconsistent metrics —both between systems and within the systems themselves. For example, one system reports processor time for its batch system, but it only reports processor time multiplied by memory size for its time sharing system.

These problems tend to make analysis difficult, and limit the applicability of analysis techniques across systems.

4. Data Analysis Tools and Techniques

Computer performance results from the interaction of a number of randomly presented demands which are satisfied by asynchronously operating resources. The result is that, even if the same job is run repeatedly in an idle system, the elapsed time to run a job varies by one or two percent. In more complex situations—with multiple jobs, on-line activities, and time-sharing systems—the variability can be overwhelming. Even reported processor time has been observed to vary by 100 percent (a doubled time) in two separate runs in the same multiprogrammed system. Conclusions based on a few samples of data are often incorrect because the data are not representative of the entire population. Careful experimental design and data analysis are required to avoid the problems caused by variability and complexity.

Experimental design is often ignored by analysts eager to collect data and show results. Unless objectives are carefully defined, irrelevant data can be collected amazingly rapidly (a reel with 1200 feet of tape is often filled in 20 minutes), and they may be collected on an unknown workload. This huge mass of data reports on events happening at very detailed levels. Relating these detailed data to decisions management must make is often difficult and sometimes impossible. For example, management's problem may be at rather a high level of aggregation like: "How much will next year's added on-line workload strain our capacity?" Detailed data on current program activity may be of little aid in answering this question.

Schwetman (in his paper in Section III.A) suggests that analysis techniques are not advancing rapidly for at least four reasons:

1. A lack of recognition of performance variability.
2. A lack of techniques and experience upon which to build.
3. A missing link between microscopic measurements and macroscopic questions.
4. A lack of load characterization methods.

5. Modeling Tools and Techniques

The two types of models primarily discussed at the Workshop were probabilistic and information-theoretic. Participants also discussed the recent increase of interest in simulation models. The upsurge of interest in both simulative and analytical modeling is partly due to a recognition that predictive capabilities need to be improved so that performance analysis can be included earlier in the computer system design process (for both hardware and software system).

Workshop participants agreed that modeling usually needs to be employed early in any phase of a system's life cycle where it will be applied. The current state of the art does not enable analysts to quickly interface an appropriate model to whatever data happens to be available, or to quickly solve very difficult problems resulting from decisions taken without the aid of performance analysis. In addition, very few models are general enough and have application techniques and caveats well enough specified that a novice can use them successfully unless he has aid from the model's developer. Some published models are currently in formulative stages where immediate application to real problems is not feasible; blind attempts to apply them are almost sure to be unsuccessful.

Several participants pointed to the limited validation work that has been done on models. Increased cooperation between people interested in measurement and those interested in modeling appears needed to improve the situation. Two of the participants indicated that they had already begun such a cooperative effort, but that far more effort in this vein is required.

6. Management

Technical personnel often feel that their management is not concerned with costs because inadequate power is given the technicians to apply their per-

formance analysis tools. In addition, they fail to demand that programmers honor standards that might dramatically improve machine performance (and often human performance) at virtually no cost to flexibility. They may fail to establish cost recovery (charging) systems to limit computer usage to jobs that are economically justifiable. They may allow vendors to dictate to them (as customers) rather than vice versa. However, the situation is seldom so simple as this.

Emphasis on computer performance may well require devotion of management effort and therefore a diversion of effort from, for example, a firm's profit-making business. To save a few hundreds of thousands, a firm might lose the opportunity to make millions.

As Seals points out in his paper in Section II.A, a lack of personnel with a proper education and the necessary imagination aggravates the situation. Management has no assurance that the set of standards and economic policies recommended by performance analysts will actually lead to improved cost-performance. The current state of the art is capable of assuring positive results when employed by good people, but performance analyses are often executed inappropriately or in inappropriate situations. Without generally applicable standards and economic policies, the domains of performance analysis' applicability are limited.

7. Research and Development

Performance measurement tools and prediction models are becoming increasingly sophisticated. Their capabilities, when applied by the people expert in using them are impressive. However, as noted above, the tools are not easily applied by a large group of potential practitioners.

Research and development is certainly required to extend capabilities beyond their current limits. However, R&D must also be devoted to making currently existing and potentially valuable tools applicable to real problems by people without several years of specialized training. Jeffery's paper in Section III.A highlights a particularly important goal in this regard: the enhancement of accounting systems and software to support analysis of their outputs.

One of the strongest recommendations made at the Workshop was to have a handbook written that could summarize the techniques for applying the existing tools. This idea was advanced at the Workshop by Jeffery (see Section III.A). In the same vein, the set of existing tools should be expanded through further development of advanced, sophisticated tools with the objective of making them useful to the vast set of problems that are presently not subjected to performance analysis. The next Section includes a more detailed enumeration of R&D activities recommended at the Workshop.

E. RECOMMENDATIONS

1. Introduction

In preparation for the recommendations activities at the Workshop, four position papers were prepared for participants to digest in advance, in the areas of Standards (by Bemer), Professional Activities (by Hamming), Research and Development (by Burrows), Education and Training (by Noe). At the Workshop, participants divided into panels to formulate recommendations which were then discussed among all participants the next day with the objective (sometimes achieved and sometimes not) of reaching a consensus position. During the Workshop two more recommendation areas were identified, and additional panels formed on the subjects of Workload Characteristics and Monitor-Register Standardization. This Section summarizes the Workshop's recommendations in these areas.

2. Standards Recommendations

Workshop participants were unanimously in favor of the following recommendations:

A representative organization such as the National Bureau of Standards (NBS), American National Standards Institute (ANSI), or Computer and Business Equipment Manufacturers Association (CBEMA) must formulate guidelines for:

1. Terminology;
2. A Minimum Set of Accounting Data;
3. Intrinsic Monitoring Capabilities for Computing Systems.

The word "guidelines" is a much weaker term than "standards." It implies a set of broadly disseminated reference definitions which the community recognizes as nominal and preferred usage, but which are not universally binding. In some areas, such as the Monitor-Register discussed below, attempts were made at the Workshop to press toward advocating standards, but unanimity could not be achieved. In general, the reluctance to advocate standards was based on a feeling that the CPE field was not sufficiently well understood yet. Standards developed at this time might not be sufficiently easy to apply, might be discriminatory, and might stifle innovation. Thus they might not be a net benefit for the field. In some areas, such as benchmark standardization, even guidelines were felt to be too ambitious.

However, there was a very strong feeling at the Workshop that significant performance losses result from the current confusion with respect to definition of such terms as CPU utilization, response time and overhead, with respect to proliferation of incompatible sets of accounting data, and with respect to definition of interfaces to monitoring tools. The recommended guidelines are needed as soon as possible. At the very least, their formulation will serve as a stimulus to improved communication in the CPE field, and in some cases they might serve as successful prototypes for an eventual standard.

3. Monitor Register

Connecting a hardware monitor to a computer system sometimes creates problems. Probe point identifications are sometimes difficult to obtain, and sometimes they do not exist. Attachment to the wrong point is easy, but detecting the problem from the resultant data is difficult. Attached probes can load circuits and cause temporary hardware malfunctioning; careless attachment can physically damage the computer. Laying cables disrupts operations as floor panels are lifted, and careful analysts often demand that computing be halted while the actual connection is performed. All this would be unnecessary if a standard attachment plug were provided.

Hardware monitors are capable of collecting data cheaply that current software monitors cannot collect easily or at all. They facilitate measurements that are independent of the host machine and therefore can be used in situations where reliability is low. In addition, these monitors can be used on different hardware and software systems so that comparisons can be made through time and across installations. Finally, communication to the machine for purposes of on-line performance enhancement is virtually impossible without some special interfacing device. While measurement through hardware is far from constituting the entire realm of performance analysis, it is important enough that mainframe vendors (and peripheral manufacturers) should recognize the user's need in this area.

The panel which met on this topic suggested that a special register be implemented for monitoring. This monitor register would be implemented differently for different hardware, and a manufacturer might choose to implement successively higher levels of the register over the lowest, level one, register. The various levels are as follows:

Level One: Lowest level register, designed to facilitate current techniques. It would consist of buffered lines to show device activity status and would have complete documentation on logical and electrical characteristics.

Level Two: A register to enable software in the host machine to communicate with the hardware monitor. It would consist of a register one word wide, loadable by the host machine's software, with half loadable from a protected state and half from an unprotected state.

Level Two (Extended): Intended to ease monitor design. It would save the unprotected half of the above mentioned word so that bit status set by a user could be maintained for that user.

Level Three: Full memory bus capability. This level would bring out (buffered) instruction address register(s), data address field(s), operation code(s), comparator status, etc.

Level Four: Communication to host system. This level would consist of a register readable (in both protected and unprotected states) by the host machine for input of special resource-allocation messages from a monitor.

The monitor register suggestion generated much discussion with some people maintaining that it assumes the current situation as the long-term technological environment. For example, micro-programmable devices throughout a system might make definition of words like "device active" impossible. Therefore, future monitoring capabilities should be

designed by manufacturers so as to provide a recommended set of data, but with complete freedom of choice as to the technology and architecture of those capabilities. Other participants argued that the results of past vendor designs for facilitating performance analyses did not indicate that users should wait to see what vendors might implement. Attempts to obtain a definitive consensus on this issue by vote were inconclusive, with many abstentions.

4. Workload Characterization

A recurrent topic during the Workshop was the necessity for better means of workload characterization, i.e., of determining meaningful categories of workload types, and parametric forms for describing a workload of a given type. Such a capability is important because it provides the necessary framework for:

1. Verifying performance improvement hypotheses by enabling an analyst to normalize performance improvements during periods of changing workload;
2. Predicting trends in computer usage of various types, and predicting the resulting resource strains;
3. Providing functional forms and parametric data to enhance analytic modeling;
4. Providing useful parameters for closed-loop monitor-scheduler modules in advanced operating systems;
5. Improving the quality and comparability of benchmarking activities.

A working group was convened at the Workshop to try to develop a definitive workload characterization. The group found it was not all that easy. As for categorization, there are several classes of categories which include at least the following:

1. Job or Transaction Characteristics
   a. Value of job completion (function of time, input data, state of application system, etc.)
   b. Resource demands
      1. By component
         (a) hardware (CPU, core, disk, channel, etc.)
         (b) software (compiler, I/O, user code, etc.)
      2. By usage pattern (probability distributions of resource demands)
         (a) by timing
         (b) by amount
2. Inter-job characteristics
   a. Dependence of job completion value on completion of other jobs
   b. Probability distributions of job interarrival times

The process of determining workload types appears to involve an intuitive cluster analysis with respect to the above categories, in order to identify clusters of jobs with similar characteristics such as student jobs, accounting jobs, I/O-bound jobs, on-line transactions, etc. Determining the appropriate parametric forms for each type generally involves a similar, but usually more quantitative analysis. Some guidelines with respect to these determinations are given below.

1. The most useful form and level of detail of a workload characterization depends on its application. This implies that workload characterization is generally an interactive, circular process.
2. A workload characterization is useful only to the extent that the necessary parametric information is easily gatherable. A good example is the two-parameter job characterization (CPU seconds of execution and kilobyte-minutes of core residence) sufficient to provide effective job scheduling in the Air Force business data processing system cited in Section I.B.4. A counter example would be a 100x100 contingent probability table of memory references in a complex Monte Carlo simulation model.
3. Workload characterizations are often machine-dependent. For example, initiator and terminator activities are quite time consuming in IBM 360 machines, but usually negligible on CDC equipment. This implies the need for extreme caution when characterizing a workload to serve as a reference for benchmark tests during the equipment selection and procurement process. In some cases, a workload characterization suitable for benchmarking may be unachievable.
4. The primary needs in the workload characterization area are for an increased level of empirical information exchange on the utility and achievability of various characterizations, and further complementary work toward an underlying theory which is relevant and accurate both in explain-

ing previous situations and in predicting future situations.

5. Professional Society Activities: Information Dissemination

Workshop participants voted 23 to 2 in favor of the following recommendation:

> The professional societies should treat this field no differently than any other. The societies should provide the usual channels of communication but should not themselves try to provide or measure compliance to standards.

The dissenters pointed out that there exist some professional societies which promote and monitor standards, and that professional societies in the computer field should exert more leadership in this direction. The majority opinion was that other types of organization (e.g., NBS, ANSI, CBEMA) were better suited for standards roles, and further that the major needs in the CPE field at this point were along the traditional professional society lines of stimulating and facilitating professional communication and information dissemination.

In the area of information dissemination, existing publication channels were considered generally adequate, as long as SHARE and ACM's Special Interest Groups on Measurement and Evaluation (SIGMETRICS) and on Simulation (SIGSIM) continue their trend toward publication of well-documented results of CPE efforts. There was some concern that professional journals in the computing field were overly biased toward publishing theoretical rather than empirical results.

One topic of particular concern was that of model validation. Users of analytic or simulation models of computer systems currently have no way of determining the extent to which the model has been validated for their situation, often leading to lost time and effort, to duplicative validation activities, and at times to inappropriate management decisions. Workshop participants felt that much could be done, within professional societies and elsewhere, to encourage and communicate the results of model validation activities. The initiatives at the February 1973 ACM SIGMETRICS Conference were a valuable first step in this direction.

6. Research and Development

Workshop participants were strongly divided on the matter of R&D priorities in the CPE field. After some discussion, it appeared that the most productive approach would be to ask participants to list their choices of the (roughly) three most important R&D projects they would fund if in a position to do so. The results, representing 23 responses, are given in Table 5 below.

TABLE 5.—*Desired R&D projects*

Theory (4 categories)	8[a]
Measures and criteria	7
Model validation and refinement	7
Workload characterization	6
National CPE laboratory	5
Closed-loop monitor-manager	5
Representation of systems and information structures	5
Comparative measurements collection	4
Hardware-software monitor	2
Variability and predictability	2
Statistical methods	1
Programmer productivity determinates	1

[a] General theory, 3; Analytic models, 3; Queuing theory, 1; Work-energy theory, 1.

Most of the entries are fairly self-explanatory, but the National CPE laboratory deserves some added explanation. It would involve the provision of a computing center stocked with measurement and evaluation tools available to theorists and experimenters wishing to test theories and hypotheses on computer system performance.

Most of the discussion of this concept centered on the problem of maintaining a representative real-world workload on an experimental system. Many users with deadlines would prefer not to use such a system even if it were available free of charge. However, it would be most valuable for the facility to run live production work, both by itself and in concert with a set of representative parameterized workload characterizations.

Other recommendations of the R&D panel drawing more general support from the Workshop participants were various information dissemination and consolidation activities such as specialized workshops and conferences on CPE theory, workload characterization, etc., channels for review and evaluation of R&D work, and reference books for the field. In this last area, participants were polled for their opinions on the

most valuable yet-unwritten document in the CPE field. The results for the nine responses received are given in Table 6 below:

TABLE 6.—*Valuable documents: Unwritten*

4—Measurement & Evaluation Handbook (when, how to use accounting data, monitors, simulators, etc.)
1—Exposé of the Unreliability of Every Known Comparative Measure
1—Facilities Management for Small-to-Medium Computer Centers
1—Organizing and Managing Measurements and Evaluation
1—Applied Statistics for Computer Performance Analysts
1—Integration of Four Types of Performance Measurement into a Single Tool (internal hardware monitors, external hardware monitors, software monitor, "mini" software monitor)

7. Education and Training

The Workshop participants strongly endorsed the recommendations of the Education and Training Panel that a coordinated program of CPE education and training should be established and supported by universities, funding agencies, large user organizations, and professional societies. The program would have two main focal points for educational leverage: education for motivation to increase awareness of the general potentials, pitfalls, and procedures of CPE; and education for competence to increase the quality and quantity of practitioners in the CPE field.

In the area of education for motivation, the main targets are:

- Managers at the intersection of authority over computing resources and computer users;
- Computer center managers;
- Lead systems programmers;
- Users;
- Vendors—hardware and software.

The most promising mechanisms for attracting and motivating the above individuals are:

- Seminars, tutorials, and workshops for focal points of purchasing control (e.g., state agency D.P. boards. professional organizations, such as the American Bankers' Association, American Management Association, and Federal agencies);
- Books and periodicals: case histories, etc.

In the education for competence area, the main targets are:

- System programmers;
- Application programmers;
- System designers;
- Direct users;
- New professionals entering the field.

The most appropriate mechanisms for attracting and motivating the above individuals are:

- University—regular students;
  —continuing education;
- Co-op programs;
- Summer courses;
- Books, periodicals;
- Professional societies—ACM, IEEE, etc.

In addition more detailed recommendations for a university CPE education program are formulated by Noe in Section IV.F. Specifically, university education in measurements and evaluation has the opportunity to do the following:

1. Spread a performance-oriented viewpoint among those preparing for teaching and practice in the field. This should orient problem solvers to attack what is important, not just what is interesting. When resources permit it, this can be extended to professionals returning to the university for "refresher" courses.
2. Stimulate research and development of means for measurement of complex computer hardware and software systems through dissemination of understanding of the problems and possibilities.
3. Influence other computer science courses so that they include measurement and evaluation viewpoints relevant to the particular topics, such as compilers, operating systems, architecture, logical design and data structures.

Toward this end, the following recommendations were developed in the university education area:

1. Measurements and evaluation viewpoints and techniques should be taught at the university level to spread consciousness of the importance of the topics, and to encourage research and development.
2. Initially this should be taught as a separate topic, through formal courses and individual studies and projects. The ultimate aim should be toward inclusion in other courses where a measurement and evaluation view is important, and

the separate course work should be needed only for advanced topics for students who specialize.

3. When taught as a separate course, measurement and evaluation should be placed at an intermediate level—after students are well aware of the functions of hardware and software systems, but before advanced courses on design of such systems.
4. Familiarity with statistical methods should be acquired through prerequisites and should only have to be reviewed as part of the measurement and evaluation course work.
5. Measurement and evaluation should be taught in conjunction with course work on modeling of computer systems.
6. The particular type of modeling (e.g., simulative or analytical) emphasized is less important than the viewpoint relating the modeling method to measurements and evaluation. The models (be they predictive or descriptive) should be used for their ability to provide evaluative information, and for their provision of a context for communications about measurements and their meaning.
7. Information should be exchanged on a continuing basis concerning the concepts to be taught and the most effective methods of conveying them. The SIGCSE Bulletin of the ACM provides one good forum for such exchange—probably better than the Performance Evaluation Review which is received by ACM SIGMETRICS members, who are already convinced of the topic's importance.
8. The most important recommendation is for continued research attention to develop the principles pertinent to measurement and evaluation of computers. This is a joint responsibility of those in industry, government and the universities.

In view of the general consensus at the Workshop that a great deal of the improvement available through CPE techniques could be achieved with present-day technology, a high priority on educational activities to unlock such a large savings potential appears well justified.

REFERENCES

[1] "Opportunity for Greater Efficiency and Savings Through the Use of Evaluation Techniques in the Federal Government's Computer Operations," U.S. General Accounting Office, Report B–115369, Aug. 22, 1972.

[2] Hall, G., and J. Wixson, ed., "Computer Measurement and Evaluation: Selected Papers from the SHARE Project," SHARE Inc., 1973.

[3] Proceedings, ACM SIGME Symposium, February 1973.

[4] Warner, C. D., "Proceedings, 1972 Fall Joint Computer Conference," pp. 959–964.

[5] Savings from Performance Monitoring, EDP Analyzer, September 1972.

[6] Gilchrist, B., and R. E. Weber, ed., "The State of the Computer Industry in the United States," AFIPS, 1973.

[7] Miller, R. B., "Response Time in Man-Computer Conversational Transactions," IBM Technical Report TR 00.1660–1, January 1968.

[8] Doherty, W., "Scheduling TSS/360 for Responsiveness," Proceedings, 1970 Fall Joint Computer Conference, pp. 97–111.

[9] Martin, J., "Design of Real-Time Computer Systems," Prentice-Hall, 1967.

[10] Yourdon, E., "Design of On-Line Computer Systems," Prentice-Hall, 1972.

[11] Williman, A. O., and C. O'Donnell, "Through the Control 'Multiprocessor' Avionics Enters the Computer Era," *Astronautics and Aeronautics*, July 1970.

[12] Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," *Datamation*, May 1973, pp. 48–59.

[13] "Analyzing Patterns of Illness," IBM Computing Report, Spring 1973, pp. 8–12.

[14] Kolence, K. W., "The Software Empiricist," *Performance Evaluation Review*," Vol. 2, No. 2 (June 1973), pp. 31–36.

[15] Morris, M. and A. Pomerantz, "Shapes Highlight Strains as Performance Plotted," *Computerworld*, October 3, 1973, pp. 13.

[16] Noe, J. D., and N. W. Runstein, "Continuous Computer Performance Monitoring," Univ. of Washington, July 1973.

[17] Burns, J., "Ford CUE Study: Six Stages of Measurement." Computer Analysts and Programmers, Ltd., London, Nov. 1973.

[18] Bell, T. E., B. W. Boehm, and R. A. Watson, "Computer Systems Performance Improvement: Framework and Initial Phases," The Rand Corporation, R–549–PR, August 1971.

[19] Quade, E. S., and W. Boucher, "Systems Analysis and Policy Planning: Applications in Defense," *American Elsevier*, 1968.

[20] Morrison, R. A., "An Approach to Performance Measurement," IBM TR 00.2272, February 1972.

CHAPTER II

CPE Objectives and Criteria

A. SPECIFIC CPE CONSIDERATIONS: SOME CASE STUDIES

The application of CPE techniques in an operational environment generally exposes a number of unsolved conflicts between management objectives and criteria at the computer subsystem level (e.g., CPU utilization), the computer system level (e.g., throughput), the end-user subsystem level (e.g., terminal-operator performance), and the end-user system level (e.g., customer satisfaction). For example, in an airline reservation system the following conflicts in objectives might occur:

- Creating a situation in which execution occurs largely from low-speed core can produce high values for CPU utilization but low throughput.
- Creating a situation in which on-line operators (e.g., airline reservation clerks) must batch their inputs can produce high values for throughput but low operator performance.
- Creating a situation in which customer options are automated (e.g., computerized seat selection for airlines) can produce high values for operator efficiency but lower customer satisfaction.

The three papers in this section discuss how CPE objectives and criteria at various levels are formulated, iterated, and applied with respect to their particular end-user area. DiNardo's paper focuses on the special problems encountered by large banks, where the financial penalties of missing daily deadlines can far outweigh the daily costs of computing. Seals discusses the evolution of CPE objectives and criteria at General Motors, as an example of a large industrial corporation with numerous autonomous divisions. Browne discusses the difficult problem of developing CPE objectives and criteria for a university computer center, which has an enormous diversity of usage patterns, priorities, and resource demand patterns with which a single facility must cope.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, Computer Performance Evaluation: Report of the 1973 NBS/ACM Workshop, March 27–30, 1973, San Diego, Calif. (Issued August 1975)

Computer System Performance Factors at Mellon Bank

George P. DiNardo

Mellon Bank, Pittsburgh, Pa. 15230

Mellon Bank has acquired or developed a series of software measurement tools which gather statistics related to job production and system utilization. No one software tool meets all the bank's needs and, on occasion, resort is made to a hardware probe. Of particular benefit have been data reduction schemes, file managers, and query languages that facilitate rapid reduction, summarization, search, and calculation of the raw measurement data. A continuing aim has been to use the same body of basic measurement data for job production statistics, system tuning data, and user costing schemes. In spite of the use of benchmarks, etc., there is concern about a basic inability to assess accurately total third generation computing capacity in a rapidly changing hardware and software environment. The increasing need and the capability to make more accurate, timely, usable, and meaningful data immediately available to the final user requires use of teleprocessing, file management, and query systems that in a narrow and traditional sense are "wasteful" of system resources on a massive scale.

Key words: Reduction of measurement data; software measurement tools; total computing capacity; user satisfaction measurement.

This paper addresses computer system performance factors as viewed by a commercial bank, Mellon Bank of Pittsburgh. This institution is the fourteenth largest bank in the nation with average daily transactions of $1.3 billion and average daily check processing which approaches one million items. In addition to providing a range of batch and on-line EDP services by both local and remote means, Mellon Bank provides automated bank services for over one hundred correspondent banks. Additionally, it provides payroll and other services for up to 250 commercial enterprises. The computing power of the bank is concentrated in an IBM Model 195, a Model 158 and a Model 65, with all tape drives, disk drives, printers, etc., connected in a shared or switchable mode. The monthly rental equivalent (most of the gear is purchased) approaches $700,000 per month.

The necessity to provide a variety of teleprocessing services with high availability, to produce thousands of batch reports with tight daily deadlines for the bank and for the others, and to provide increasing facilities for direct access of data by the final users, forces the bank to concentrate on computer system performance evaluation in terms of reliability, availability, and recovery. In addition, there are the considerations related to return item deadlines and loss of float. The legal implications of missed deadlines to the Federal Reserve could cause the bank to have to honor (pay) an item which should be dishonored due to the loss of the "Right of Return." The financial implications of the loss of float through late or improper processing in a bank of our size is staggering. The daily average transit amounts to $300 million, with a possible loss of $290.00 per day per million dollars worth of checks that are not processed in a timely fashion. Approximately one billion dollars in funds are transferred in and out of our bank daily, exposing us to litigation for loss of float if the funds are late in arriving at their destination. All this reinforces this concentration on reliability, as opposed to resource optimization.

The first several paragraphs describe the mechanics

of the software measurement techniques now employed by the bank.

IBM's System Management Facility (SMF) produces 47 records on job and step utilization of system resources (core, device allocation, EXCP's). EXCP's can be analyzed to produce utilization by device, control, unit, and channel. Excluded under current SMF records are data on system tasks and their utilization of resources. This includes the basic control program plus spooling programs such as IBM's HASP. Thus, SMF is a good source of data for job production and cost accounting. Additionally, such gross indications as IPL's, accesses to test libraries and various utilities, bottleneck reports, etc., can be produced from SMF. SMF is the primary tool for benchmark tests which are based on job production. It is also useful for data set activity analysis. SMF places an overhead burden on the system of less than five percent. The following additional software and hardware measurement facilities are employed for analysis and improvement of system tasks:

A. GTF traces all loads, attaches and links. Useful for tuning for residency of SVC's, etc.

B. TFLOW is similar to GTF in that it captures all interrupts. Good for detailed tuning, e.g., start I/O loops on ITEL disk devices was a problem which TFLOW helped to trace.

C. Above two are so voluminous that it is necessary to use FORTRAN & AL programs to reduce data on them, and to produce selective summaries.

D. TSTRACE—A TSO driver oriented tracer. Approximately 47 entry codes provide data on swap loads, core occupancy, SVC fetch time, number of swaps/terminal interaction, and overall system response time. TSOPAP is used to reduce TSTRACE to cumulative percentages.

E. OSPT1—IBM program product fills the gaps in SMF by highlighting system utilization of channels, control units, devices, and core. Helps in study of shared DASD problems and SVCLIB utilization and cross-system contention.

F. LEAP—Time distribution of execution by module, down to individual instruction. Pinpoints I/O bound problems in applications.

G. System LEAP—Similar to OSPT1 but enables analysis disk measurements such as cylinder crossed, etc.

H. SUPERMON—Largely replaced by System LEAP and OSPT1 in our environment.

I. AMAP—A hard trace (not a sample) of system workload which can be projected or interpolated to show execution date on other theoretical hardware configurations.

J. SMI—A hardware probe which when used as a supplement to above software measurement techniques, is particularly useful in measuring cross device utilization or, in other words, for recording waits by one system for device being used by another system.

K. MAIL—Mellon All Inclusive Library—Used primarily to store and release a large series of jobs which simulates a live operating environment. Includes our live operational jobs and test jobs. The mix can be adjusted to provide desired ratios of compiles, sorts, etc.

L. Human Observation—Our operators are trained to record observations of system slowdown and provide extensive documentation of console status and other indicators. They also invoke such dumps as DYNADUMP for snapshot of a running system and RESDUMP for a stopped system (lockout or hard loop).

M. Console logs are valuable analytical tools and produce a chronology of internal and externally induced events which indicate system performance or lack of it (DA's, DU's, DQ's, and DSQA's are particularly useful).

N. The Analysis File in our Inquiry System is an an example of an application which has a built-in capability for recording events, transactions, etc., which often are used in trouble and performance analysis.

Many of the preceding produce voluminous data which must be reduced by summarization of some kind. GTF and TFLOW are examples. We have prepared FORTRAN and Assembler reduction programs which are used to key on particular events, devices, tasks, etc., and then produce meaningful totals, ratios, distributions, etc. Other measurement devices have a capability of self-reduction. An example is LEAP which decreases the sampling ratio as measurement time increases.

IBM's Generalized Information System is an example of a free form query, macro-level file management system. It is used to additionally summarize,

profile, and compute ratios on a total of all or selected items, times, programs, etc. It is used to produce periodic job production analysis, test job throughput, device utilization, etc. It is very useful in special studies to monitor long-term availability of particular devices, e.g., it was used to pinpoint a particular tape drive that, over an extended period, had a higher than normal rate of failures.

GIS is also useful in experimenting with various computer costing methods. A variety of computer costing methodologies can be manipulated to determine the effect on key production applications.

It is difficult to over emphasize the importance of an ability for programmers and non-programmers to have a file management/query language capability for summarization and manipulation of production and performance data. The ability to analyze iteratively relevant data on specific problems is of equal importance to more comprehensive, complex formulae which present overall indicators. That conclusion may not be as true in a non-commercial environment. The need here is for sufficient capacity to cope with hourly deadlines; changing weekly, monthly and seasonal workloads; vagaries of weather; hardware problems; communication difficulties; and power outages.

Added to above is the capability to invoke such facilities in an on-line mode. The ability to input queries via terminal and receive output is particularly useful in critical production and performance problems. It also accelerates the iterative process described above. In our installation, the capability is provided by high speed remote job entry, by query capability inherent in the on-line system, and by internal time sharing. Interactive compilers supplement the time sharing facility.

The widely appreciated phenomena of the difficulty of assessing capacity in third generation equipment is, of course, felt in this bank. It impacts planning for new equipment, configuration of existing equipment, creation of production schedules, benchmark comparisons of existing and proposed equipment, etc. It becomes a particular problem in determining capacity for use in any computer costing scheme and, particularly, in a system based on full cost recovery.

We have found no completely scientific or even reasonably precise approach to estimate a full capacity base. The changing mix of jobs, job priorities, equipment, and software all contribute to the difficulty. Even if we were able to determine current capacity, there is the difficulty of assessing capacity for a base period such as a year.

An example will serve to highlight the difficulty. An important element of a component costing scheme is main memory. In an MVT system, there is great difficulty in forecasting the effectiveness of the operating system in managing main storage for both system and applications. Core fragmentation is a continuing problem which changes in magnitude continually as job mixes and priorities vary. Further, during any one year the economics of main storage are changing.

Further, during 1972 any one application could have run on IBM supplied Model 65 main storage, on leased Model 65 main storage from two sources at two different prices, on extended core from Ampex on a Model 50 or on a Model 65 (at two different prices and two different speeds), on a Model 50 in IBM supplied main storage, or on a Model 195. Throughout the year, the mix varied from month to month. Much the same story could be told for processors, tapes, disks, channels, etc.

We realize that, in spite of those difficulties, it is theoretically possible, given sufficient expertise and time, to arrive at a reasonably precise estimate of capacity which represents the entire year. However, the resources needed to accomplish that are considerable and there are invariably many other demands on the talent required to accomplish such a total or, more precisely, to continue accomplishing it at frequent intervals. The dilemma is that without such a base of comparison, precise and detailed actual measurements of utilization are of limited value.

Thus we have convinced ourselves, or rationalized if you prefer, that a more reasonable approach is one based on brief historical measurements of production and utilization, on manufacturer benchmarks and other estimates, and on our own benchmark data. Such consideration of relatively limited quantitative data is mixed liberally with "educated" guesses of reasonable capacity and, at times, supplemented with examination of astrological phenomena.

With the reintroduction of virtual storage and virtual machines by IBM, we expect that our difficulties in this regard will increase substantially. We recognize that, initially, our system will be in a relatively untuned state. Just as with our present software, extensive resources will be devoted to tuning hardware, control software, and applications. The tuning

process will be a continuing one as hardware, releases, measurement capability and tuning capability change. The point is that as technology provides improved performance and economics, there will not necessarily be a concomitant increase in our capability to assess capacity and measure performance accurately in very large multi-programming and multi-processing installations. There are many indications that the opposite is true from both a theoretical and a practical point of view.

The resources devoted to system measurement and tuning at Mellon Bank are estimated to be the equivalent of two full-time personnel. However, it is not the full-time task of any one individual. In an effort to represent the interests of the many organizational elements concerned with performance priorities, and to coordinate the many special skills involved, a special tuning committee has been formed. It is chaired by the head of the Systems Programming Section and has membership from Data Processing Operations and Teleprocessing Coordination Units and from Application Programming on-line systems, functional systems, and training sections. The group meets weekly and based on specific management or self-imposed projects, assigns measurement tasks to individual members, analyzes results, and makes recommendations for control software, application program, and hardware changes.

On an almost daily basis, we are reminded that there is another aspect of computer performance evaluation which is implied or affected by quantified performance evaluation only in the most indirect sense. This is in the satisfaction of the end user. At first glance, this appears to be too obvious to mention. Yet we are discovering that user satisfaction has an increasing number of facets which are taking on ever more complex interrelationships.

It is obvious that unless the automated application produces the required reports in the time required, a basic user dissatisfaction will result. However, even where the user needs are perfectly understood by the analyst and programmers, some compromises are almost certain to result in the form, timeliness, accuracy, etc., of the final data. This underlying discontent has and will continue to lead to greatly expanded memory, disk, line, and terminal facilities, to more powerful and flexible processors, and to greatly expanded control and application software.

Thus, we believe one of the most important contributors to complexities in hardware and software design, and, consequently, to its measurement has been the pressure, real or imagined, by users upon EDP practitioners and EDP designers for ever more complex compilers, utilities, teleprocessing controllers, data base managers, and query languages. As the per execution and per byte costs have declined and the human costs have risen, we have become more and more willing to accept macros, modules, and even larger elements of software which are far less efficient and more generalized than lower level languages.

Those pressures, have, in turn, led to increasing investments by EDP suppliers and independent software houses in ever more effective and complex offerings which frequently are even more voracious in their consumption of system resources. This phenomenon is well recognized and needs no further elaboration. What must be understood, however, is the effect of pressure by top management and end users alike for effective, useful, and available systems in addition to pressures for economy in EDP costs. It is our observation that in progressive and reasonably successful enterprises, the former pressures are often greater and more frequently expressed than the latter, economic pressures.

Once an enterprise begins to depend on an on-line system for its daily operations, the pressures that develop when that system falters are far more constant, powerful, and multi-directional than those from the comptroller and top management. It is for those reasons that we have given much more attention to computer system performance evaluation that is directed toward availability and effectiveness rather than toward efficiency.

The creation and implementation of file management, query languages, time sharing, data base management, data entry, and on-line file inquiry systems have consumed a great deal of our software development resources for many years. Today, those systems and control system support of them consume more main storage space and CPU cycles than do standard batch application. Such systems, because they impact final users so quickly and directly, are usually more effective in producing user dissatisfaction and complaint than batch systems where hardware and software delays may be more recoverable.

In our view, the ability of inquiry, time sharing, file management, query, and similar systems to make data more directly available to final users is a key to their justification of such systems. If, with little training and experience, the final user can interatively

retrieve, peruse, summarize, and calculate basic data elements, then we can justify usage "wastage" of system resources at a level which would have horrified EDP practitioners a few years ago.

In recent years, we have frequently made that trade-off in the interests of reduced system development time, reduced human costs, increased final user participation, and increased on-line and even real-time capability. We recognize the difference between making such a choice and being aware of the costs and with making the choice with little appreciation of the wastage that can result. We quickly admit to a lack of precision in such matters. We have made some attempts at quantifying the developmental and operational costs of, for example, an application written in COBOL versus the same one written in a high level query language. We do attempt to determine the costs of an application in batch versus on-line, of data entry versus card batch, or remote job entry versus local batch, etc. Nevertheless, we are not precisely aware in every case of the extra cost in system usage of user-satisfying but costly facilities nor are we always sure that such "advanced" developments are using resources as efficiently as possible.

The point we are making is we believe that many traditional schemes of computer system measurement and evaluation may indeed be missing the point. Such systems, we suspect, concentrate on the micro aspects of measurement and miss the macro impact of the effectiveness, satisfaction, and availability of the system to the final user. Such approaches can lead to over concentration on bits and bytes which results in comparatively well-tuned systems that do not provide the facilities and capabilities demanded by the firm's competitive situation. Ideally, we should be both penny wise and pound wise but, unfortunately, our capabilities and tools to be pound wise in a precise fashion are very limited.

In summary, Mellon Bank has acquired or developed a series of software measurement tools which gather statistics related to job production and system utilization. We have found that no one software tool meets all our needs and, in addition, resort to occasional use of a hardware probe. Of particular benefit to us have been the data reduction schemes, file managers, and query languages that enable us to reduce, summarize, search, and calculate the measurement data. Our continuing aim has been to use the same body of basic measurement data for job production statistics, system tuning data, and user costing schemes. We are concerned about our basic inability to assess accurately our total capacity in a rapidly changing hardware and software environment. We attempt via benchmarks and the like, to make such assessments, but freely supplement such data with horseback guesses. We are also mindful of the increasing need as well as the capability to make more accurate, timely, usable, and meaningful data immediately available to the final user. Such need and capability, we suspect, is increasingly our raison d'etre, yet we recognize that, at least today, such systems in a traditional sense are wasteful of system resources on a massive scale. And, finally, we recognize how poor is our capability and opportunity to measure the performance of these splendid systems in which we take so much pride.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Computer Performance Analysis: Industry Needs

Eugene Seals *

The Rand Corporation, Santa Monica, Calif. 90406

The computer has grown in its ability to provide information processing support to the automobile industry. The budget for computers and related items has also grown, both in magnitude and as a percentage of the expense of sales. Computer Performance Evaluation (CPE) has begun to offer help in increasing the productivity of EDP equipment. The auto industry has recognized the need for CPE and engages in CPE programs with vigor. The general CPE consciousness taking hold within the EDP community promises benefits to the auto industry, which currently is involved with generating its own CPE talent and techniques.

This paper discusses the CPE needs of the automobile industry with specific references to General Motors where the author was previously involved in CPE activities.

Key words: Computer performance evaluation (CPE); CPE policy; CPE education; CPE imagination; CPE instrumentation; EDP productivity.

1. Introduction

In 1972, General Motors produced 7,800,000 cars and trucks worldwide. The company has significant computer hardware at 38 locations. Its related computer hardware costs have been estimated at over $50 million per year, or over $6 per vehicle. Compared to other overall General Motors financial performance figures in 1972, computer hardware costs were thus over 0.2 percent of total sales, over 2.6 percent of net profits, and over 18 cents per share of common stock. Thus Computer Performance Evaluation (CPE) activities can have a significant leverage on overall corporate financial performance, if they can reduce the overall hardware expenditure without adversely affecting productivity.

Corporate management recognized this in setting up a corporate-level Machine Efficiency Project in 1970 to develop and apply CPE techniques. However, widespread CPE application has been slow to come because of the autonomy of the various GM Divisions; the lines of authority above a Division and above the Machine Efficiency Project intersect at the Chairman of the Board. The only corporate management checkpoint on the use of CPE techniques is during the approval cycle for new hardware procurements, and it is at this point that CPE activities make the most headway in many of the Divisions. The situation appears to be similar in other large United States automotive corporations, and often for other large multidivision industries.

Despite its size and the generally conservative public posture of the corporation, General Motors is far from being staid and inflexible. EDP, for instance, is the subject of intensive study and in 1972 bore little resemblance to 1970 EDP. Predictably, a number of the installations have grown in computing power. At the same time, some locations have begun to pool their resources into consolidated centers, while other installations are absorbing workload formerly processed by smaller machines in other parts of the corporation.

This paper summarizes the types of decisions made within GM on computer hardware and the related performance criteria considered most important for computer systems. It then discusses the major CPE needs

* The views presented in this paper are those of the author, based on his experience in the automobile industry, and do not necessarily reflect the views of the General Motors Corporation.

of GM and similar companies; these tend to be in the areas of policy, education, imagination, and instrumentation.

2. Decision Categories

In the auto industry, management is concerned with four classes of decisions. These involve the critical areas of (1) capabilities, (2) capacity, (3) control, and (4) cost.

Capabilities—What capabilities shall I provide? What compilers, utilities, technical support, hardware, etc. will facilitate attainment of my organization's objectives?

Capacity—How much of each capability should I provide?

Control—How do I police the system, the coders, the users, in order to insure meeting organizational objectives, given the information systems workload, the hardware-software system, and the people involved at any moment in time?

Cost—What is the most cost-effective approach to meeting the objectives of the computer department?

3. Performance Factors

The following discussion is presented from the point of view of those installations already in existence.

When data on utilization of capabilities (such as compilers) are collected, conditioned, reduced, and summarized, management is in a position to know what use is being made of capabilities which are currently provided. Such data can also be instructive in projecting utilization of other capabilities which may be in the planning stage. Further, simple displays of such data over time puts management in a position to respond to trends which may be developing in demand for given capabilities.

Similarly, reports summarizing utilization of the computer system and its various subsystems can give management a feel for aggregate demand for resources. Since averages and summaries tend to mask certain other important attributes of utilization, management will find distributions showing utilization of each component to be helpful. In fact, without such distributions many important conclusions and decisions may be ill founded. This is not to say that distributions are needed in all cases. Averages and summaries also provide information which is vital to effective control of resource utilization. Often two or more dissimilar subsystems may provide similar service to users. Users may have a preference for one subsystem and thus provide heavy demands on that resource while the other resource goes underutilized. Having hard data on such a situation, management is in a position to take appropriate action. Where such disproportionate utilizations are deemed good, management may want to procure additional capacity in the one area while decreasing capacity of the other resource. On the other hand, management may wish to take steps to encourage more balanced utilization of each resource. In such a case, the billing system is a convenient mechanism for implementing this kind of management decision. By adjusting the differential in cost of the resources to users, management encourages greater utilization of the underutilized component and to some extent penalizes user decisions to overutilize the other resource.

Part of the effective management of data processing is keeping costs in line. Once-simple decisions have become complicated as the result of the proliferation of products in the marketplace which can supply the information processing requirements of the auto industry. IBM, Honeywell, Control Data, Burroughs, and Univac represent the manufacturers of medium-to-large computer systems. Varian, Digital Equipment, Texas Instruments, and Data General are illustrative of the many manufacturers of small and special-purpose computing systems. The manufacturers of components which take the same (or equivalent) approach to a given problem are numerous, not to mention vendors who compete on the basis of plug-compatible equipment. As superior technology becomes less dominant in the decision to buy hardware, vendors are competing more in the areas of cost and vendor support. In order to make decisions which minimize total cost to the organization, management needs not only cost data, but also performance data on the various offerings which propose to satisfy given information system processing requirements.

There are several sources of performance data, each of which may be more appropriate for certain classes of decisions than others. First of all, the vendors supply specifications which describe the raw performance attributes of their components and systems. In addition, your technical staff may collect performance data under live or representative operating conditions. You may prefer in some instances to simulate the interaction of your requirements with the ability of

the proposed system to deliver. Or you may want to inquire of other installations who have investigated performance of a given product or product class, given the large investment which may be required to keep abreast of all developments in the data processing industry. Whatever route is chosen, management is still left with the analysis question. Are the advantages/disadvantages of one system enough to justify the cost differential? Or vice versa?

Whereas distributions may be required in order to know the impact of utilization and demand for certain resources, gross measures are adequate for certain other needs. Average CPU time and average I/O counts per job step, for example, are good management indicators. Trends in each of these areas have implications for management decisions. Where remote users are concerned, for example, management needs to know levels and trends in demand for each remote terminal and for each user of a shared remote station. Without such data management must make decisions blindly regarding deployment of remote terminals.

Given the fact that the automobile industry has a very substantial inventory of computer hardware, software, and applications in use at this time, current efforts are to a very great extent determined by present components. However, the same questions are important across vendor lines. Some vendors provide accounting systems and monitors to collect performance data. To this extent, one vendor's systems may be superior to another's. On the other hand, since the ability to implement monitors and accounting routines has been demonstrated for virtually every major system, the existence of such tools may not be an overriding consideration in choosing a system. Vital performance areas such as response time and turnaround time retain their mandatory time boundaries whether a given user is dealing with a 360/67 or with some other piece of hardware. On the other hand, it is quite evident that different users (and different uses) may have widely differing performance requirements. Some analysis is required (be it ever so little) in order to ascertain the boundaries for each user application.

4. Needs of the Automobile Industry

CPE needs of the auto industry are similar to those of other users. CPE needs fall into four categories: policy, education, imagination, and instrumentation.

4.1. Policy

In general, corporate, middle, and installation managements neither request nor receive reports which measure the productivity of computer systems in the auto industry. There are some notable exceptions, of course. In the main, however, performance is regarded as a binary question: Was the work delivered on time, or was it not? As long as the work gets done on schedule, management has little incentive to be concerned with performance. As each division of General Motors, for example, is highly autonomous, the Corporate Information Systems and Communications Activity group is not in a position to control the divisions relative to performance measurement. A director of data processing operates so as to minimize his maximum regret. The penalty for carrying excess, underutilized capacity is less severe than the penalty for being late with crucial reports. There is apparently a fairly widespread assumption that the penalty is severe for not meeting schedules for ALL reports. Of course, this is true to some extent. But few, if any, managers have conducted a systematic investigation to assess the costs incurred when individual reports are delivered some minutes or hours after they are normally delivered.

A Comptroller's Circular Letter was a major step forward in the communications of General Motor's desire to increase the productivity of its EDP investment. In addition, the Corporate Information Systems group involved the divisions in writing performance measurement and evaluation guidelines into the Corporate Data Processing Practices and Procedures manual.

4.2. Education

Most of General Motors' installations are still in the process of developing resident performance analysts—a core of individuals who plan and implement aggressive, on-going, productive performance improvement projects within their installations. The rate at which this development is taking place leads one to be concerned that the turnover rate exceeds the rate of development, hence defeating the project.

Furthermore, few computer science curricula provide students with the opportunity to concentrate in the area of computer performance measurement and evaluation. We addressed this lack by 1) conducting

a performance evaluation seminar and 2) organizing a Machine Efficiency Committee which met bimonthly to exchange information relative to progress in the installations. Task forces worked on projects conceived by the committee.

For several years now, UCLA has offered short courses which emphasize the importance of CPE, serving as somewhat of a leader to other major universities. Examination of recent college schedules reveals that other universities have begun to add CPE courses to their catalogs. The University of Michigan now offers a fairly substantial number of courses in performance analysis. The University of Washington, Brigham Young University, the University of Texas at Austin, and others are offering courses in this area, too.

Professional associations such as SHARE and GUIDE have very active projects which deal with performance evaluation. Other associations such as ACM and CUBE have begun to organize similar groups within their respective organizations. In fact, the impetus has now carried overseas, receiving attention from the British Computer Society and the IBM users' group SEAS. The National Computer Conference (and its precursors) also encourages work in CPE.

Some computer manufacturers have issued statements which indicate their intention to allocate more resources toward gaining a better understanding of the performance of their systems. In addition, vendors of measurement tools provide a great deal of training in evaluation techniques in the interest of creating and exploiting a market for their products.

An interim solution to the pressing need for education and training may be the establishment of continuing "institutes" in performance evaluation within the larger auto companies. Either the Corporate data processing staff or the General Motors Institute could provide the facilities and the required instruction within GM. Dr. Thomas E. Bell has conjectured that it takes approximately 24 months to train a performance analyst. Estimates from other sources are even higher. It becomes imperative, therefore, that some action be taken without further delay.

Perhaps equally important, if not more so, in the area of education is the cultural lag which exists in top and middle managements regarding performance evaluation. The computer has been a real blessing in the control of the large plants and divisions of the far-flung operation of the automobile industry. The computer is responsible for such large cost reductions, cost avoidances, and new opportunities for revenue that management is reluctant to question the efficiency of the "goose that laid the golden egg." Even when management casually opens the subject, computer technicians are generally prepared to illustrate how effective a tool the computer has been to the business. Managements will have to recognize the merits of a performance evaluation activity in their shops before the gains which some companies report will be realized in the automobile industry.

In addition, many EDP professionals have absolutely no performance consciousness. Until this obstacle is overcome, little progress is likely to be made in this vital area. Of the many fine data processing installations within General Motors, two stood out as early leaders in performance measurement and evaluation. One shop emphasized configuration and operating system tuning and monitored its system daily. In this same shop, however, the COBOL programmers noted that their applications tended to be I/O limited. The program monitor reminded them of this fact without presenting any solutions. On the other hand, the FORTRAN programmers were not part of the computer organization and—because of fairly liberal budget allowances for computation—were not motivated to seek to generate highly efficient programs nor to perturb currently operating programs.

The other highly aggressive shop had done a better job of selling performance evaluation to its management and had a more balanced program of utilizing configuration monitors, program monitors, accounting data, and other tools. A great deal of progress has been made over the past year. A number of managers and performance analysts have accepted transfers within General Motors and have begun to instill an interest in performance evaluation in their new organizations.

4.3. Imagination

The discipline of computer performance measurement and evaluation is still somewhat fledgling. In addition, techniques are not as readily transferable as some might wish for them to be. Consequently, it is necessary for each performance analyst to utilize, not only his skills relative to the computer, but also his creativity in order to identify and take advantage of opportunities which may be installation unique.

4.4. Instrumentation

From the homespun routines of the self-starting programmer to the half duplex system which monitors its other half, computer performance measurement and evaluation has made considerable progress, especially in the last five years. Yet, today there still is no comprehensive, integrated package which will supply the answers needed to maximize computer system performance. Packages such as PPE and CUE are complementary; but correlating the outputs of one with the other is a fairly major exercise.

A comprehensive package might consume an inordinate amount of scarce resources when used. But this is true only when packages are developed in the conventional manner. Retrofitting is not known to be an especially efficient engineering technique. Performance monitors are essentially retrofits. What is needed is an operating system with built-in software probes which can be activated quickly and easily and which consume few scarce system resources when in operation.

IBM recently unveiled her new operating system for virtual System/370 with no plans for built-in performance measurement beyond that available on System/360. (There is reason to believe, however, that not all the capabilities of the hardware and its measurement subsystems have been announced to the public yet.) Some vendors of performance measuring software have announced that their products will be upgraded to deal with the new operating system. At best we may expect better retrofits than was true of first generation software monitors now that the monitor developers have gained some experience.

Computer performance evaluation has indicated a real need, not just for performance analysis tools, but for efficient vendor-supplied software as well. Utilizing the available techniques, General Motors, in various locations, tuned the vendor-supplied sort and spool writers. Other locations, in responding to urgent screams from on-line users, were able to uncover critical areas of inefficiency in vendor-supplied on-line support packages. All this has highlighted the need for more efficient vendor-supplied software. In the interim, alternative sources have been selected for some utilities. Other replacement utilities are also being evaluated.

Another area in which there is a real scarcity of tools and expertise is the real-time, on-line teleprocessing applications area. There is a rather distinct trend toward utilizing and relying on more and more on-line applications in critical decision-making and resource-utilization, market-responding areas of the automobile business. Time is very precious to the auto companies as they strive and excel in world markets. Products specifically designed to monitor and evaluate teleprocessing applications are typically retrofitted, and, therefore, inefficient in their interfaces with the computer system and its workload.

At General Motors we saw a rather distinct need for a reduction in the manual intervention required to optimize the hardware and software systems and the application programs. Several products have begun to emerge which offer promise in this area. CAPEX automated the optimization of COBOL object code. Boole and Babbage attempted to automate the tuning of the operating system.

Hardware monitors are excellent tools to utilize in evaluating a computer system. The major drawback to the use of hardware monitors is the high probability that something will go wrong, particularly in selecting and connecting probes to capture desired signals. Several attempts have been made to facilitate verifying that probe points are properly selected and connected. Some practitioners utilize an IBM Field Engineer program to exercise the system in a predictable fashion, during which the connections can be verified. Other users have written their own exercisers. Boole and Babbage suggested the permanent connection of their probes to commonly used pins and the labeling of the connections so that subsequent monitoring sessions would not require reverification of the connections in great detail. The Computer Measurement and Evaluation Project of SHARE has suggested that mainframe manufacturers provide a standardized hub for easy access to the signals most often monitored. IBM provides a limited hub on the 360/85 and the 370/165.

At General Motors and at many other places, there exists a need for continuous monitoring of the computer system. Traditional monitoring studies have endeavored to select representative periods during which the system would be monitored. Modifications made on the basis of extrapolations from such sessions have a high probability of being effective for total system efficiency even though some individual jobs may suffer. Monitors have tended to produce a single figure which they have offered as being true in an average sense for the period monitored. Some disadvantages of this approach include the fact that

averages may conceal important information regarding the distribution of two or more populations and their interactions across time. Boole and Babbage attempted to answer this need with a fairly inexpensive monitor. The University of Texas incorporated software probes (i.e., hooks) into their CDC 6600–6400 operating system. Tesdata Corporation is now offering a tiny, very inexpensive hardware monitor.

Because of its size, General Motors is in a position to develop its own operating system. There is a need for an in-house operating system which is tailored to the purpose for every company which buys a computer. If GM builds its own OS, it can build out these inefficiencies and build in aids such as software probes. The GM/OS would be self-regulating, reviewing demands on an hourly basis to determine, for example whether the resident SVC list should be modified dynamically, or the RAM, or BLDL, or the size of SQS, or the size of the supervisor itself. Dynamic modifications would take place without operator or programmer intervention. So far, corporate management has declined to apply resources in this direction. The valid business reason is that General Motors is not in the software business. It is felt that managerial and technical expertise can be used to bring in greater returns in the mainstream of the business.

5. Summary

The computer has grown in its ability to provide information processing support to the auto industry. The budget for computers and related items has also grown, both in magnitude and as a percentage of the expense of sales. Computer performance evaluation (CPE) has begun to offer help in increasing the productivity of EDP equipment. General Motors has recognized the need for CPE and encourages the autonomous divisions to pursue CPE programs with vigor. Despite early start-up problems, a CPE-consciousness is taking hold within the EDP community and promises results over the near term. While several options remain open to the auto industry, it is gratifying to see the increased formal preparation being offered in computer science curricula in the leading universities. This increased awareness will provide CPE a prominent position in the attitude with which EDP professionals approach their work, reducing the need for CPE to be a separate, unpopular activity.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Performance Factors For University Computer Facilities

J. C. Browne

The University of Texas, Austin, Texas 78712

The unique problems of performance evaluation in a university computer facility arise because of the diversity of usage patterns and resource demand patterns with which a single facility must cope. In such an environment, one key element of performance evaluation is the characterization of the workload; another is that a high premium must be placed upon adaptability and flexibility of the management algorithm. Subsystem performance can normally be subdivided into the service given the different classes of users. It is a balanced and adequate performance in these subsystems which is basically the most important criterion in universities.

Certain performance factors are discussed, such as end-user satisfaction, resource utilization, and usage of system subcomponents. The skewed competence of users, peculiar to a university environment, is also discussed as a factor affecting performance.

Key words: End-user satisfaction, performance evaluation, performance factors, resource utilization, user competence.

1. Introduction

The unique problems of performance evaluation in a university computer facility arise because of the enormous diversity of usage patterns and resource demand patterns with which a single facility must cope. The university is typically a microcosm of the commercial and professional world. The types of computing which are done at universities include all of the traditional categories: 1) very heavy research computing, which demands enormous consumption of resources by a single job; 2) very high volume batch (low resource consumption per job) utilization from student users in computer supplemented instruction and computer science courses; 3) large interactive usage resulting from research which again typically consumes substantial quantities of resources per job; 4) high volume remote terminal interactive usage from students in such functions as computer-assisted instruction. This is essentially transaction oriented usage where rapid response is essential; 5) real-time experiment control. This is usually a fairly minor function in universities, however, except with special purpose dedicated systems; 6) information retrieval and analysis on large data bases. The most significant measures of performance are the user satisfaction variables in a weighted average over all classes of usage. Interference of the sub-systems one with another in competition for resources, particularly memory resources, is often a limiting performance factor.

A special problem with universities is that the user population tends to be highly transient with an enormously skewed distribution of competence. The distribution typically has a median at a very low value. These factors lead to certain specific effects upon both the external management policies and the internal management policies of the computer system. Clearly, the internal management must be sufficiently flexible so as to rationalize conflicting and competing demands arising from different patterns of resource utilization by different categories of work. The system itself must be sufficiently simple in operation that even an uninformed and ignorant user can use its resources reasonably efficiently for a simple task.

Another characterization is that university computer centers typically have fluctuating patterns of

demand. Typically, there will be a daily cycle (which is common in commercial installations as well) but also a cycle extending over several months representing semester or quarter terminations.

In an environment as diverse as this a key element of performance evaluation is the characterization of the workload. Secondly, a high premium must be placed upon adaptability and flexibility of the management algorithms. This is true not only because of the fluctuation of demand on the regular and known cycles and the diversity of usage, but because universities are seldom in a position to reflect changing circumstances by capital expenditure and must instead change operating policies of the system in order to cope with changing environments.

Subsystem performance for university computer facilities can normally be subdivided into the service given the different classes of users, the heavy research users, the student batch users, interactive service, etc. It is a balanced and adequate performance in these subsystems which is basically the most important criterion in universities. Total usage patterns are indeed very significant. Reaction to performance measures is usually made in response to an imbalance which reflects itself in poor performance by one or more of the subsystems.

Absolute system performance is often an extremely important element in university computer systems since it is generally the case that there is a paucity of resources to be spread over an enormity of demands. Only by extremely efficient utilization of the total system, can it be hoped to spread the resources adequately to cope with the diversity of needs.

The discussion of performance factors which follows is not precisely tailored to the suggested categories of the other sessions but caters to their existence as far as was reasonably possible.

2. End-User Satisfaction

The evaluation and importance of end-user satisfaction in a university environment is complicated by the complexity of that environment, just as are other measures. We can, for the purposes of present discussion, divide the usage into four functional categories. There is research batch computing, which is characterized by very large programs consuming large quantities of resources, either CPU, random access memory, or I/O channel transfer time, or all three. There is high volume batch associated generally with instructional purposes. These usually arise from scheduled classes which assign problems to be solved on the computer. These jobs are usually characterized by each individual unit being relatively modest consumers of resources, but the aggregate will frequently amount to the numerical majority of programs run on the system. Very often these programs will make use to a very high extent of standard system software components, and involve relatively little execution of user written code, perhaps 5 percent or less.

The interactive usage too can usually be categorized into two factors. There will be instructional interactive usage which will very often be of the form of computer-assisted instruction, tutorials, problem solving sessions from terminals, etc.; the resources consumption per unit of this type of program tends to be quite small and its usage of systems software tends to be concentrated around a few specialized components. Research interactive computing is of an entirely different nature. It tends to be programs which have very large field length requirements, very often manipulates very large data bases, and may, although not necessarily, require considerable CPU service.

Clearly, the measures of service for each of these groups is entirely different. The large scale research user is primarily interested in the amount of resources dedicated to his project over a period like a day, a week, or a month. The urgency for deadlines and the interest in turn-around time arises here primarily in the debugging phase or when graduate students must finish theses and dissertations, or when papers must be prepared to meet deadlines (as is usually the case). To the high volume small batch user the most important criterion by far is very rapid turn-around time. It is often the case in universities that there are stations available where students can submit their jobs and (hopefully) wait for them to be returned without undertaking other activities, i.e., a laboratory environment where the computer can be regarded as an instrument upon which results are to be demonstrated. Turn-around to the order of 5 to 15 minutes is very significant for this type of usage. For instructional interactive work, by far the highest criterion is stability. Very often the student is assigned access to a terminal or has available the usage of a terminal over a relatively restricted period. If the system is not stable and does not provide service at approximately the expected level, he will have difficulty maintaining the necessary work pace. Therefore, stability

is the most important element with response time of a fairly adequate level, necessary although this is not usually a critical factor. To the research interactive user, typically response time is the most important variable. This also tends to be the measure which has the most fluctuation. When a relatively small number of this class of jobs is occupying the resources of the machine, typically they will receive good response time. When, however, they are competing with a large number of student users, response time suffers and they typically tend to get off the system or to complain vociferously.

Each group in the university must be prepared to tolerate variations in the quality of service available to it. There are cyclic patterns of demand where students in the instructional program tend to have work clustering upon them toward the end of the semester with term projects due, back assignments incomplete which must be terminated by the semester, etc. Typically then the high-volume small-job batch load rises to peak periods as the semester nears an end. Consequently the research user finds his service deteriorating and his turn-around time becoming markedly poor.

The interactive usage, on the other hand, tends to have a more daily cycle. In the early mornings, typically one will find research users dominating the resource consumption on the interactive system. In midday it is typically the case that a large number of classes will be scheduled and many terminals will be occupied by students. In this circumstance one sees the workload changing dramatically in character to where performance is dominated by the existence and demands of a large number of students operating from terminals.

The factor which must be kept in mind is that each group of users is capable of swamping and dominating the system during its peak periods. The management factors which affect the performance of each must be such as to assure each group a reasonable share of resources, while at the same time catering to the inevitable swings and fluctuations in cycles and demands which result from the structured pattern of a university calendar.

3. Resource Utilization by Classification and by Software System

A two-way classification of resource utilization is the key information in managing the performance of a university computer installation. By this I mean, the resource utilization characteristics in terms of CPU utilization, I/O channel utilization, remote storage utilization, loading functions, etc., for major system components and for user programs. The utilization of the major system components must then be classified according to the definable user groups, for example, the large batch, high-volume batch, instructional interactive and research interactive as discussed previously. This type of knowledge enables one to do effective predictive scheduling and to design algorithms taking advantage of known resource utilization patterns. It should be clear from the discussion given earlier that it is necessary to carry out measurements on a fairly frequent basis, for example, hourly during the day at least on some measures, in order to cover the daily cycles, while long-term averages are frequently sufficient for other measures.

It is generally the case that there are distinct software system elements associated with operation in each mode of usage. Typically the high-volume batch user of resource utilization will be heavily keyed to user written programs originally coded in FORTRAN (or COBOL) source. He will probably make heavy use of specialized utilities such as random access I/O managers and elements of the mathematical function library. The resource utilization of such programs is not immediately characterizable because there will frequently be a high variability with the programs alternating from a strongly CPU bound phase to perhaps a strongly I/O bound phase, etc. The high volume small batch users, however, typically make very great utilization of the system's compilers and utilities executing very little user written code. This means that their resource utilization characteristics can be established by examination of the mix of system components which they have. It is generally the case that the system components will have fairly fixed and discernible patterns. For example, system utilities such as copying of files from disk to disk, the FORTRAN compiler, the MIX simulator, etc.. will have fairly constant patterns. These patterns may or may not be favorable in terms of utilization of all-over system resources. In many cases, however, if their characteristics are known, their modes of usage and characteristics can be altered to improve all-over system performance. A very detailed study of the usage of system components, compilers, utilities. etc., was carried out on the University of Texas system

by D. S. Johnson.[1] The research interactive computing typically executes its own code (or interpreter) and system utility handlers such as editors, data management packages, etc. The instructional interactive usage typically has its resource consumption dominated by requests for services to the operating system. Very often the proportion of resources consumed by the operating system in the university computer system will be directly proportional to the number of instructional interactive users on the system.

Summary: Workload characterization decomposing the work on the system into software system utilization and then into functional classification is the key element for controlling and balancing the performance of a university computer facility.

4. Skewed Competence of the Users and Performance

The enormously skewed degree of competence of the users of a university system propose a unique problem. The system must be so designed that elementary decisions lead to near optimal patterns of resource utilization. Simple acts of carelessness can cause severe degradation. For example, consider the typical high-volume student FORTRAN compilation. The FORTRAN compiler will occupy perhaps 24K words of storage, while the student program will typically occupy only 4K to 8K. Unless the system automatically reacts to provide field length reduction, then it may be that a student job will go through the system occupying for its entire lifetime the field length determined by the compiler. In a system such as that represented by the University of Texas at Austin, where the demand for compilation and execution of student jobs rises to the level of 5,000 to 6,000 a day, this represents a very significant utilization factor.

Other examples come from usage of the interactive system. The interactive system at the University of Texas is extremely permissive. It will allow virtually any system to be invoked from the interactive terminal that can be invoked via the batch system. It is sometimes the case that a student will attempt to use a program such as the MIX simulator from a terminal. As is usually the case with most interpreters, the MIX system has a typically long CPU run-time to accomplish a relatively small amount of work. When it is being operated from a terminal, competing with other interactive programs, this relatively long field length interpreter will attempt to acquire several seconds of CPU service at a shot. This results in the large field length occupied by the ssytem being swapped in and out of the core a large number of times to accomplish a small amount of work, thus performance can be significantly affected by an uninformed user invoking improperly a system component which performs effectively when operated in the proper environment.

5. Usage of System Subcomponents

The utilization of hardware system subcomponents is of interest in university environments from two viewpoints. It is frequently the case that the shifting workload will move the bottleneck in the system from one area to another. For example, the research interactive users typically run into a bottleneck with respect to available core storage. The instructional interactive utilization typically finds as the bottleneck the swapping capability of the system. Thus competitive and cyclic effects are the most interesting aspects for the all over performance analysis with respect to subcomponents utilization.

It is also frequently the case, however, that a university will have a research program involved with performance measurement and utilization.

In addition, it is also frequently the case that a university computer center will have available to it a group of systems programmers who can be used on a daily or weekly basis for system-tuning. Thus university environments are likely to have more use, on a day-to-day basis, for information relating to subcomponent performance than are typical industrial and commercial shops.

[1] Johnson, D.S., Ph.D. dissertation, The University of Texas at Austin, 1972.

B. GENERAL CPE CONSIDERATIONS

The papers in the previous section took a vertical slice through the CPE objectives and criteria at several levels of management concern, as seen by a single type of organization (banking, automotive industry, university). The papers in this Section take a horizontal slice, across many types of organizations, at a single level of management concern, while also attempting to relate the primary objectives and criteria at their level to those at neighboring levels.

Gary Carlson's paper discusses CPE objectives and criteria at the **computer subsystem** level, primarily concerned with such items as CPU utilization, I/O channel overlap, compiler speed, and the like. Morrison treats the **computer system** level, primarily concerned with such items as throughput, response time, and overhead. Kiviat discusses the **end-user subsystem** level, generally concerned with transactions processed per day or per employee. Nielsen's paper concludes by discussing objectives and criteria at the **end-user system** level (profit, stability, customer satisfaction, etc.), and their relation to objectives and criteria at the other levels.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Computer Sub-System Performance

Dr. Gary Carlson

Brigham Young University, Provo, Utah 84601

The goal of performance measurement is to improve the performance of the system and reduce the cost. The present measurement tools start at the computer subsystem level. A thorough understanding of these tools seems to he necessary before we can move beyond the subsystem level into the overall system, then the computer operations, then the computer management, and hopefully beyond. Subsystem measurements have a direct impact on equipment configurations in terms of reduction of presently installed equipment, postponement of planned equipment, selection of new equipment and comparison between different vendors. How to detect and interpret low and high device utilization and uneven distributions of activities is covered.

Overhead is explored in several contexts and an attempt to generalize the concept of overhead is made. Performance comparisons are made between theoretical and practical maximums. Suboptimization is discussed, pointing out that some suboptimization can have no bad side effects and should be achieved.

Variability in present measurement techniques is bothersome and is discussed briefly. A need for better reporting techniques is indicated.

Key Words: Distribution; low utilization; overhead; suboptimization; subsystem measurement; variability.

1. Introduction

The goal of most computer performance measurement is to improve the performance of the system and reduce the costs. The particular goals to be achieved may differ from installation to installation, but all are seeking for an improvement of performance at lower cost. The measurement tools that we have today generally start at the computer subsystem level. These tools allow us to measure time and events and to record these measurements on magnetic tape for further analysis. The use of these tools can help us to clarify some of the basic measurements that are possible in computer systems.

Hopefully, such measurements, if they can become precise, consistent, and agreed upon, could lead to new measurement techniques to measure more than the subsystems in the computer room. We might later evolve ways to measure the effectiveness of the people in the computer room, then of the computer operation within the institution, and perhaps the effectiveness of the institution within society. This may seem like a long way to go, but it seems worthwhile to try to aim in that direction. The first task for us, then, is to improve our techniques and understanding of measurement at the computer subsystem level.

2. Impact on Equipment Configuration

Measurement efforts today are often designed to help us achieve a more optimum configuration of equipment. This optimization can be achieved in the following ways:

A. Reduction of presently installed equipment.
B. Postponement of planned equipment.
C. Selection of new equipment.
D. Comparison of different manufacturers equipment.

All of these techniques have been used to improve performance and reduce costs. No one should be exclusively considered since all can be considered at the same time. The above aspects of computer performance are usually involved in the "tuning" of a computer system. Tuning may be the first necessary step along the path to an optimum performing computer installation.

3. Factors to Interpret For Management Action

It is becoming easier and easier to obtain raw data from probe points or software programs on events that occur with a system. The interpretation of this data has not become appreciably easier over the few short years of the performance measurement activity. Often those involved in obtaining the maesurements are not in a position to directly influence management decisions that need to be made to make any necessary changes. We need to develop ways of training competent computer people to become competent interpreters of this increasingly large mass of performance measurement data. We also need to train such people to not only interpret the data, but also interpret the data in such a way that those responsible people in management can understand what corrective action should be taken. One possible way of looking at this kind of data is suggested in the following outline:

Low Utilization. Low Utilization is often a serious problem that is worth attacking directly. Low utilization can lead to excessive costs though it does give good performance to the users. Low utilization has at least three dimensions:

Low device utilization. This is a case where a given device such as a tape, disk, card reader, printer, etc., has a very low overall utilization. This condition can usually be detected with simple monitoring techniques. Such things as stop watches or reading meters or just ilstening to the sound of the printer can give appropriate clues. More precise measurements clearly indicate the low device utilization. Most installations seem to have an exponential curve for the activity on a number of similiar devices. For example, if we have six tapes on a single channel, the first two tapes have 60 percent of the usage, the next two have 30 percent and the rest are spread out over the remaining devices. This seems to hold for tapes or disks. With this kind of distribution we can often remove the lowest used device with little degradation of performance. Similar conditions are found with channels. If we have multiple channels on a system, their activity often follows a decreasing exponential curve with the implication being that if you have the possibility of reducing the channels and thus reducing expense, it should be seriously considered.

Low CPU activity can be caused by too little work, or as Schwartz [1] [1] has shown on a B–6700, too much work. On this virtual memory machine a high work load causes thrashing, with high I/O activity resulting in the CPU waiting for I/O. In one experimental overload the CPU activity dropped from 90 percent to less than 30 percent!

Low device overlap. In an idealized computer system one could imagine that every device in the system would be running concurrently. This would mean everything running in parallel. In actual practice we never even come close to this possible ideal. We have the myth in the industry of overlapped operations, but measurement has shown that fact is often different from myth. The most startling lack of overlap is the disk seeks on the 2314 type devices. In repeated measurements made on IBM 360/30's, 40's, 50's, 65's, and 75's a typical percent of time that there are two or more seeks in process is around 1 percent! It is possible that systems could be changed to improve this overlap and would lead to improved performance. Another instance of the lack of overlap is in channels which have tapes on one and disks on another. Often the overlap between channels is very low, on the order of 4 to 5 percent. It would seem that improving the percent of overlap would improve performance. One trick that seems to work is to split some tapes and some disks onto each channel. This does require a two channel switch, but can pay good dividends.

Low device availability. Low device availability will force low utilization. Monitoring will indicate that low utilization is occurring. Operating procedures may be one cause. The scheduling of the work load can cause apparent peaks of activity and then lack of availability during these times with subsequent periods of enforced idleness.

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

Equipment failure also contributes to device unavailability, but this can easily be monitored and high failure rates should never be tolerated.

High utilization can often lead to problems of poor customer service though they may make operational people feel good in the sense of a high usage of the equipment. There seem to be at least two areas of concern here.

Utilization can be artifically high due to the connecting of asynchronous devices. One of the most common examples of this is the attachment of bulk core that opeartes at a slower cycle time than the CPU to which it is attached. In one measurement of the IBM LCS on a Model 50 where the LCS cycle time is 8 micro-seconds and the CPU cycle time is 2 micro-seconds, it was found that 30 percent of the time was spent with the CPU in a completely stopped mode waiting for synchronization with the LCS. During this time the CPU appears to have very high utilization. This is true whether measured with a vendor supplied meter or wait light or CPU active pin. Software monitors also record the artifically high usage figure. During this state of apparent high usage the CPU is unable to do productive work and can cause serious bottlenecks in terms of total throughput in the system.

Excessively high utilization can appear good from the operations standpoint but can lead to abominable service from the user standpoint. There seems to be an inherent conflict that to achieve high CPU activity we need to have a reasonably long queue of input jobs. The very fact of having long input queues means that we will be giving poor service to the customer. Each installation must try to achieve that balance of customer service and equipment utilization that best satisfies the particular organization's goals.

Uneven distributions of activity often lead to performance anomalies. These distributions often are of two forms:

External which seems to be related to the scheduling of jobs into the computer system. If little effort is made to schedule jobs, we often find a normal poisson curve describing the interval between arrivals into the system. This always causes peak activity and consequent overload on the system and degradation of service. Scheduling should be encouraged which will flatten out the distribution of inter-job arrival times. To the extent that users can be made aware of this problem and adopt this as part of their scheduling algorithm they can achieve better service on the same equipment.

Internal uneven distributions of activity often occur. A system that does not have the I/O and CPU balanced will exhibit this behavior. Activity may peak on particular channels or particular devices and in one sense we then chase a peak load around through the system, with the attendant lack of parallel processing and lack of throughput. The internal uneven distributions of activity should be attacked by careful analysis of concurrent activity and efforts to adjust the operating system as well as the hardware to allow more even flow. We need to have greater consideration of a computer system as a continuous flow process similar to a refinery rather than as a discrete series of independent events. In other words, the pipelines allowing information flow should be matched in diameter of capacity.

Overhead is an often used but rarely defined term. Management may ask about overhead and we say it is too high, but we lack good definitions of what overhead is or should be or could be. One of the desperate needs in the performance measurement field is clarification of the term overhead or at least attempts at an operational definition. Overhead is considered to be those activities that occur in a system that do not directly contribute to the achievement of the activity we are trying to do. We should realize that overhead is not restricted to the computer room but seems to occur in all activities of life. In fact, one of the interesting kinds of comparisons is that if we assume that the average human sleeps approximately eight hours and spends approximately two hours in eating and tending other biological functions, we find that the average human spends 10 hours out of every 24 hours in what might be called overhead functions. This comes out to a somewhat surprising 42 percent overhead figure for the average human. Buildings have approximately 40 percent of the interior space consumed in what might be called overhead functions such as stairways, hallways, storage closets and so on. With this thought in mind we should not be too shocked if there is some overhead that seems to be impos-

sible to reduce. Overhead in a computer system can be found in at least four different areas:

The operating system overhead. This is one which has received many tirades in the literature but little light. Those of us who were running on computers before operating systems arrived recall the great skepticism we had, fearful that the overhead would be an intolerable expense. I think many of us also recall the great surprise we had that in spite of the machine cycles given to the operating system we got a significantly greater number of jobs through the system. I recall in the installation I was working at when the operating system arrived, we had a three day backlog of work to be done. When the operating system was installed, we got caught up by mid-afternoon of the first day. Virtual systems may be similar. It would seem that we need serious scientific study to clarify the general concept of overhead in computer systems; but also to relate this to the larger general phenomena in any system whether man-made or living. In an attempt to start on this, we have teased out one aspect of what might be called overhead—though part of it is certainly a necessary evil. On an IBM 360/50 running under OS MVT 20.6, we ran a job stream with SMF, or the facilities accounting routine, imbedded in the system; measured the precise time with a hardware monitor; and then ran the identical job stream through with the SMF extracted from the system. The CPU time was 13 percent less with SMF removed. The elapsed time was 17 percent less with SMF removed. One might then assume that SMF as part of an overhead function may be taking as high as 13 percent of our available cycles. One needs to then carefully ask the question if this is the kind of trade-off that you want for your installation.

Operator overhead can come from operational inefficiencies where operators do not anticipate the demands of the system and spend time searching for tapes, paper, instructions, people, etc. Performance measurement can show when the system is waiting. It could seem that with an ideal situation the human opeartors could be completely paralleled with the computer system. This would mean that the system would give the operators messages far enough in advance that they could take whatever human action is necessary. Many of the software monitors give useful information in terms of device-not-ready which can indicate some kind of operator problem, therefore, possible overhead.

Compilers always involve overhead. Since compiling is a translation from the human to the machine language, that operation in itself does not directly contribute to getting the job done. It is true that writing in higher level languages can save time; and, therefore, this kind of overhead can be useful—though we must realize that this overhead exists and is measurable. The compiler programs also need tuning.

Applications programs may have overhead, but the overhead is generally significantly less than the items listed above. Applications programs should be carefully screened for excessive statements or inefficiently coded sections of the program. Software program analyzers should be used on all high usage production programs. Many COBOL analyzers have been useful in reducing core used by 20 percent and reducing time by 10 percent to 20 percent.

Performance Comparisons. There is a need for comparisons of actual performance measurements. In the early days of performance measurement, just a few years ago, a person could make a measurement and not know if it was good or bad. Today many people may think they have a good measure, but may be only comparing it to other measurements made in their particular installation and not realize what other installations have been able to achieve. Performance comparisons can be four different kinds:

Theoretical versus Actual. The theoretical possible performance of a device or system is often different from the actual. It seems that this problem can be further broken down into the following areas:

a. The average actual measure. This would be a range of measurement that people have found in different installations, and this would not say whether these were good or bad, just a fact of observed measurements. This will be discussed later.

b. The maximum theoretical possible is 100 percent activity for all devices. 100 percent CPU busy is both a theoretical and an achievable figure. One could say theoretically that a

printer or card reader could be driven at 100 percent activity. This, however, does not square with reality.

c. The maximum practical possible is often unknown, but is knowable. A typical example is with card readers, punches and printers where it always takes some time to change paper forms or ribbons or cards. We hope to soon be reporting the results of a study where we try to drive some of these devices at the maximum speed and allow normal operational servicing and changing of forms, ribbons, etc. It appears now that the maximum practical possible activity of EAM type devices is rarely above 90 percent. It also appears that the maximum practical activity on a disk channel may be around 50 percent and on a tape channel around 40 percent. Accurate measurements of these maximum practical limits could help us gain a clearer insight into when there are conflicts within a computer system. If these maximum practical values were known, it might also be possible to make a start on more automatic analysis routines. Such routines could take performance measurement data and compare it with these realistic maximums and by some kind of realistic analysis give suggestions to operational management of the most pressing problems to be solved. Schwartz and Wyner [2] have reported a monitor with built-in threshold values. When CPU activity exceeds 75 percent for more than 5 minutes, a record is made.

d. There appears to be some optimum utilization figure for each particular installation. This optimum is determined by a large number of variables which are often in conflict. In our particular installation in the University, we pushed the equipment and load to a sustained 90 percent CPU activity on a Model 50. We found that this led to very poor customer service in terms of response time and turnaround time. We then set about trying to improve the capacity of the system, change the operational procedures, scheduling, algorithms, etc., to reduce this activity. We now have an installation goal to come down to 65 percent CPU activity. We feel that this level of CPU activity will allow us a proper return on equipment while providing acceptable service to the customer. Each installation needs to carefully consider what its optimum performance level might be.

We need to have available performance comparisons with the rated specifications. This applies to I/O devices. We have measured some installations where the I/O equipment has been performing 18 percent below the manufacturer's rated specification. The vendor should make the equipment perform properly. Another aspect of this which is now used at Brigham Young University is in our request for bids. We specify that I/O equipment must perform to the vendor specifications as measured by our monitoring equipment, or the vendor must remove the product at no penalty to the University. This is a clean area of performance measurement since the rated specifications are always available and the measurements to determine the device's performance are easily made. The action to be taken is also clear—the equipment should be tuned to perform up to, or beyond, the rated specifications.

We should have performance comparisons with other installations. In an attempt to make a start on this, Table 1 is enclosed giving the range of measures that have been observed in various IBM 360 installations. In the table are some hodgepodge of different kinds including university, industrial, commercial, etc. These figures are crude and ambiguous due to various settings. However, they start to give some range of expected measures. Tables like this should be refined and a consistent effort be funded to obtain coherent, meaningful data. In gathering this information from various installations monitored, it has appeared possible that there may be classes of installations; for example, university settings may differ from a credit card installation. These may differ from a banking installation, these from an insurance, these from a manufacturing and so on. It would seem that if such classifications could be derived from actual data, we could have ranges of observed measures in, maybe, 8 to 10 kinds of installations. A given installation could then compare itself to the appropriate comparable kind of installation. Such information might reduce the apparent variability we see in

measures as we go from one installation to the next.

The desired level of performance is the last comparison that should always be made. Based on the goals and intent of management, we should be able to establish the preferred level of activity for the installation as a whole and then break this down to the devices. Performance measurement could then compare how an installation is doing in relation to these preferred levels of activity. It can usually be seen that higher levels of activity lead to poor service and poor customer satisfaction, and activity below these preferred levels leads to excessive costs. We need studies to help management set these desired levels in relation to the other constraints they have.

4. Suboptimization

Suboptimization is another problem in performance measurement where some people have been fearful of suboptimizing a part of the system to the detriment of the overall. It seems that suboptimization can be broken down into three areas of activity:

A. Suboptimization with no bad side effects. In this case we should clearly take whatever steps are necessary to achieve this suboptimization. In case the reader doubts that such cases could ever occur, let me mention a few. If we find that I/O gear is not performing up to its rated specifications, adjustments must be made. This can always be done during weekends or slack times with no serious detriment on the throughput of the entire system. The benefit is always good and has never had a negative impact. Another example is when we have the choice of having parts of the operating system on disk or in core. Proper analysis of the relative activity can determine which should be where. In most cases, we can reduce the core space with little increase in disk space and achieve a significant improvement in system performance. Other examples may abound and should be put into the common literature to allow all to achieve these kinds of performance improvements without requiring extensive measurement activity.

B. Some suboptimization can have some benefits, but also some detrimental effects. Here we must be very cautious and take what we can, but be alert to possible detrimental effects in the overall system. One example that can occur here is the question of faster disks. Sometimes we can put faster disks on a given channel and cause channel imbalance or channel capacity to be exceeded which then can cause a system to take extra cycles to recover. In many cases in this category of suboptimization, we are not solving a problem, but merely chasing it around the system. These should only be considered for action when we feel we can reasonably chase the problem out of the system so there are no detrimental effects at higher levels.

C. Some suboptimized effort can cause major increases in "cost" at other levels. These classes of events should be avoided. An example that has already been alluded to is to optimize CPU activity to achieve a 90 percent to 95 percent busy, but this results in terrible service to the customer. In this category we must be very careful to ask this same kind of question for the entire computer operation within an organization. We must no longer necessarily assume that anything computer is good.

5. Variability

Variability in measurement has been and continues to be a troublesome problem. Anyone performing measurements has been frustrated by the apparent instability of the same measure at different times. It seems that when we have apparent variability, this should be taken as a clear indication that we do not understand the thing that we are measuring. The more clearly we can define what we are measuring, the more stable our measurements can become. We should not be discouraged by variability in a computer system and feel that it is hopeless to try to make sense out of it. Many fields of science, even in physics, have certain side effects that are always present in making any measurement. The variability introduced by these effects can be partially controlled and those that cannot, can be analyzed statistically to improve our understanding of the underlying events and their causes. In performance measurement we must consider at least the following areas:

A. Confirming measurements must always be taken. In too many cases we have seen measurements that appear stable, become unstable with a

second measurement. This relates to the standard scientific procedure of replication of experiments. If we cannot replicate the experiment and get comparable measurements, we do not have a science, but only a fairy tale. When we measure subsystem performance and recommend action to be taken, we must always have confirming measures. To the extent that these measurements can be made with different tools—so much the better. At least we must be able to replicate the experiment with the same tool at different times.

B. We must be careful to separate fact from fiction when considering what things might be variable. For years the industry has had the common impression that peak loads always occur in business data processing installations. Assumptions have been made that the peak load demands on the equipment are significantly greater than during the "normal" processing times. Repeated measures in a number of installations show that this is not an overall phenomenon, but that peak loads in business data processing settings generally relate only to the I/O devices. Typically, CPU activity will increase perhaps 3 to 4 percent from normal to peak, where I/O may double. This most often occurs with printing devices where the apparent noise and confusion increase significantly at the "peak load" times. Variability of measurement could be clearly tied to the cycle of work to be done and thus explain and reduce the apparent variability. It is interesting to note that some measurements are constant in light or peak load, like lines or cards per minute.

C. We need to account for variability in more than a statistical sense. Some common horse sense questioning of the operational people can tell you that the chief operator was sick, or the equipment broke down, or a particular channel was malfunctioning, or the system's programmer made a significant change in device assignments, etc. To understand performance measurement data we need to be like a detective and try to uncover all related events that could help us understand what happened.

D. Variability in reported results can be due to different bases of measurement. An example is time. We need to share with each other the time base that we are using since this is so critical in all computer performance measurement. The kinds of time that are used are wall clock time, CPU time as measured by some internal clock within the CPU, systems meters, and separate clocks in hardware monitors. It does not seem practical at this time to try to urge any standards on anyone, but the least we can do is to clearly define which time we have used in expressing our measurements. This can reduce some of the apparent variability.

6. Need For Better Reporting

Much of the measurement work done today is done on a semi-crash basis to try to quickly justify the acquisition of the measurement tool and the jobs of those who are making the measurements. We find no fault with this approach, but we should try to encourage and provide inducements for people to share the results of measurements that they have made. In order to share results, we need to clarify some concepts:

A. We need basic, consistent units of measure that can have meaning from one installation to another and from one manufacturer's equipment to another. We should at least be able to give operational definitions that are consistent. We seem to have emerging defacto units of measure like CPU activity. On many machines this can now be stated as the time measured by a hardware monitor while attached to a given pin at a given location in a given machine. Similarly, software monitors can say they measure CPU activity by examining the contents of a timer register at the beginning and end of the program run. Attempts should be made to pull together measurements that have become defacto standards and make these available to the computing community.

B. In addition to better reporting, we need to work towards the concept of exception reporting. This would mean that after certain measurements have been made we could tell if these are within expected ranges or not. These relate to the data given in Table 1 where we have observed ranges of measurement. Exception reporting can only be made meaningful if we have adequate stability of measurement.

TABLE 1.—*Typical range of hardware monitor measurements as % of wall clock time*

	360/30	360/40	360/44	360/50	360/65
System Meter	21–80	56–100	83–88	93–100	90–98
CPU Busy	16–51	17–52	32–48	44–85	32–93
CPU Only Busy	16–34	15–43		21–42	
Protect Key O	19–24	18–27		38–62	
Supervisor State	20–32	22–67		30–53	25–32
Problem State		20–47	2.5–4.5	33–40	
CPU Wait On Printer	8–10	6–17	5–26		
CPU Wait On Disk Seek	3–9	2–52	3–11	1.1–13.8	
Multiplexor Busy	0.8–2.1	0.5–1.2	1.0	0.8–1.4	1.9–3.1
Multiplexor Meter In		10–26	61	40–100	
Selector Channel (tape & disk)	16–27		21–35	31–49	13–44
Selector Channel (tape)	3–15	2–23		9–13	
Selector Channel (disk)	5–19	11–18		25–39	11–48
Selector Channel (teleprocessing)					
Printer Busy	8–22	7–36	18–27	29–53	8–57
Reader Busy	3–16	5–16		11–21	
Cards per Minute	919–1033				
Lines per Minute Rated 1100	813–1077	936–1074	959	975–1140	
Lines per Minute Rated 600				570–683	
Emulate Mode	0–6		21–34		
LCS Inhibit				16–30	6–36

C. Derived variables must be considered as well as directly measured variables. Rates, like terminal response time or disk access times, are examples of indirect measures. This kind of measurement can lead to clarification of what happens during events that are of importance to people. In one study we have analyzed the events occurring within a 360/50 system during a terminal response time. The disk activity (including the disk controller) only caused 14 percent of the delay in response time; whereas, the non-interruptable processing within the operating system accounted for 64 percent of the delay. This measurement of response time is a derived measure. This particular case implies that faster disks have relatively little impact on terminal response time; whereas, certain changes that might be made in the operating system could have a significant impact. This study was indicated by a factor analysis we ran over a wide range of measurement of events that occurred during the response time delay.

To summarize, computer subsystem performance is still in its infancy and we should not be disappointed at our lack of precision or lack of understanding. It seems that we may be like a pre-Galilean lying on the grass counting the stars. If we count carefully enough we may find that there are patterns to their movement and significance in their spatial location. The instruments we have are undoubtedly not what we need in the long run, but I think we have been given a basic telescope so we can start to make more accurate observations. It appears that we still need a lot of people to help make star charts before we launch off in our space ship. We should try to improve our ability to share experience so that we can indeed work towards a science and build on each other's work.

7. References

[1] Schwartz, Jack M., B6700 "Performance Measurement and Evaluation," printed by the Federal Reserve Bank of New York, September, 1972. To be submitted for NJCC, June, 1973.

[2] Schwart, Jack M. and Donald S. Wyner, "Use of the Spasm Software Monitor to Evaluate the Performance of the Burroughs B6700," printed by Federal Reserve Bank of New York, February, 1973.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Computer System Performance Factors—Their Changing Requirements

Robert L. Morrison

International Business Machines Corporation, Poughkeepsie, N.Y. 12602

Advances in programming and computer system architecture are causing additional performance factors to be identified faster than the relationships among them are being understood. This must necessitate changes in the technology, terminology, and methodology used to measure and describe computer system performance in the future. Problems and limitations resulting from using several measures of performance popular today, and suggestions for meeting tomorrow's needs, are noted in this paper. Insights gained from evaluating IBM's recently delivered virtual storage systems during their development and early release stages provide the basis for discussing these changing requirements.

Key words: Computer system; factors; measurement; parameters; performance; performance terminology; predictability; requirements; variables; workload.

1. Introduction

We may be describing the performance of data processing systems a few years from now in ways different from how we do so today. This paper reviews the current use of several performance factors. Problems and trends resulting from today's large and increasing number of factors are noted, and suggestions for changes to meet tomorrow's needs are offered. Insights gained from evaluating IBM's recently delivered virtual storage systems during their development and early release stages provide the basis for discussing these changing requirements.

2. Some Current Practices—and Problems

A myriad of interrelated factors account for the performance of a data processing system.

Factor: agent, something that actively contributes to a result.[1]

Parameter: a characteristic element; also: factor.[1] a variable that is given a constant value for a specific purpose or process.[2]

Variable: a quantity that can assume any of a given set of values.[2]

[1] The New Merriam-Webster Pocket Dictionary (1971), G&C Merriam Co.
[2] American National Standard (ANS) Definition, ANS Vocabulary for Information Processing (1970), ANS Institute, Inc.

The terms "parameter" and "variable" are sometimes used to shade the meaning of "factor": 'variable' to connote a "given" (for example, the size of the user's main storage), 'parameter' to refer to a value selected by the user (such as the level of multiprogramming), and "factor" to mean the measure of performance, such as CPU utilization or throughput.

2.1. System and Workload Factors

It is convenient to group performance factors into two broad complementary categories:

1. System factors—those hardware and software configuration variables and operational parameters which describe the data processing system whose performance is under consideration. This group includes the hardware resource utilization factors used to create profiles of overall system performance (see Figure 1). It also includes factors which provide more detailed knowledge of how the resources are used than the cumulative totals shown in a system performance profile. One such measure growing in popularity, which combines the count of instructions executed with their timing, is the rate at which instructions are executed, termed "MIPS" (millions of

instructions executed per second). The MIPS rate of a CPU is not an engineering constant, but rather a function of the internal architecture of the CPU. It varies with the memory size, the I/O device configuration, and the instruction mix, that is, the workload being processed. Another useful feature is the number of instructions required to process a transaction. These factors are being used to classify interactive workloads for estimating response times and the maximum number of terminals or transaction rates a system can support. System factors cover a wide range of stability, size, and relative dependence on one another.

2. Workload factors—those characteristics describing the load being applied to that system. Examples of factors in this group are the number of transactions being processed per unit of time, the level of multiprogramming, instruction sequences and frequencies of use, and the number of instructions between branch instructions. Parameters describing system loading are used for such purposes as annotating measured results and representing workloads as input to models. Some workload characteristics (e.g., the number of data sets accessed) are static, while others (e.g., the working set size) are dynamic. Kimbleton [1] [3] and Lynch [2] each stress the importance of load characterization for performance prediction. These parameters vary even more than the system factors as to their level of detail, relevance, and underlying assumptions.

More useful than simply listing several factors found significant to performance, therefore, is to note some problems and limitations their use imposes, particularly in light of our experience with virtual storage systems.

2.2. Experience with Virtual Storage Systems

Improved performance in the broad sense of increased productivity is one of the main advantages claimed for virtual storage (VS) systems. A concern of the designers and implementers of these systems is to provide this capability without introducing excessive overhead. Because of this concern, as well as the lack of recognized measures of productivity, the extensive investment in performance during the development of the VS systems was oriented around the traditional throughput, response time, and resource usage studies.

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

The initial decision to implement VS systems for S/370 was founded in these beliefs and later confirmed by measurements: (a) with existing multiprogrammed operating systems, much of main storage was unused much of the time; (b) improved throughput would result if the utilization of the CPU and memory were brought into better balance, that is, were more overlapped; and (c) in most instances CPU utilization could safely be increased by the amount estimated as needed to bring this balancing about.

Almost 3 years before the August 1972 announcement of IBM's VS systems, studies of address referencing patterns using traces of several programs had been made, resulting in the now-familiar steep working set ("paracore" curves (see Figure 2).

The notion of "optimum real storage" (ORS) has been introduced now that main storage is included along with the CPU and I/O as one of the resources to bring into balanced use. ORS, also referred to by some as "paracore" or "working set size", is the number of real storage page frames required by a program in order for it to execute without increasing the page fault rate to the point of thrashing—devoting more and more resources to paging and less and less to productive work. The amount by which the real storage required to process a workload in systems without the dynamic address translation (DAT) facility exceeds the amount of real storage available for paging the workload in VS systems is known as the overcommitment of main storage. Interesting as ORS is, we so far lack a theory to explain its relationship to certain variables in a multiprogramming environment—such as main storage overcommitment, the level of multiprogramming, paging rate, and of course, elapsed time or throughput. We do know that individual programs optimized for VS systems exhibit a more pronounced knee of the curve in Figure 2. The knee is at that amount of real storage which is the optimum relative to the amount of paging activity. The system is sensitive to this ORS; that it, with only slightly less storage available, the system will thrash. It is also known that just as the ORS of individual programs varies over time, so, too, does the amount of paging at the system level vary, clustering at times of peak activity such as job-to-job transition. Thus, there must be a relationship between ORS, contention for main storage, and performance. This seems analogous to the knowledge of stand-alone CPU and elapsed times of jobs which are then multiprogrammed.

They are necessary but insufficient factors with which to predict elapsed jobstream time or throughput.

Measured data and analysis were used to show that for the size systems to be supported by VS/2, fewer page faults, though not the minimum memory size, would result from using 4K rather than 2K pages. However, in the size systems to be supported by VS/1 and DOS/VS, the number of page frames required to be available for paging allowed for only 2K per page frame. One consideration in the decision to proceed with 2K pages for the latter systems was that theoretically, up to eight pages might have to be referenced during the processing of one instruction.

The page size and other basic decisions relative to VS were being made during the time that S/370s without the DAT feature were first becoming available to the program development groups. Since there is greater credence in measured than simulation results, various means were considered for obtaining and using measured data. No single method of analysis was utilized to the exclusion of all others. A prototype of VS/2 Release 1 was built to run on the S/360 model 67, the only computer in the S/360 line with the DAT feature; a simulation model of VS/2 Release 2 was written in CSS, analytic models of VS/1 and VS/2 were used in several studies. In general, while analytic and simulation models provided information to help make or confirm several major decisions, measured data was involved in each such decision to the extent it could be obtained.

CCW translation time, the MIPS rate, paging activity and storage contention were the focus of intense study, though in the early stages paging activity and storage contention were necessarily inputs to, rather than results from, simulations. Models provided early indications that the amount of real storage available for paging would be one of the most significant factors affecting performance in VS systems. It was also known early that CPU utilization would be increased and that the page fault rate was critical to the success of the systems, since CPU time is consumed in servicing page faults.

We also recognized that introducing the use of VS and the DAT facility might have significant operational and performance consequences for an installation, and that these would differ from one installation to another based on a combination of several variables. Two results of this concern were: 1) inclusion of a generalized trace facility as part of VS, which performs tracing and timestamping of user-selected system events; and 2) studies of ways in which applications programmers can achieve improved program performance now that they no longer need be constrained to using only that amount of memory physically installed in the computer; e.g., see Reference [3].

It is useful to summarize the experience of several system performance evaluation projects not by referring to their specific technical results but, rather, by mentioning some problems limiting the usefulness of these efforts, and then noting some trends and changes in the way evaluations are or should be proceeding, in order to overcome those limitations.

2.3. Problems and Limitations

Pervasive problems related to but extending beyond technological ones are:

1. Performance evaluation can be an enormously expensive undertaking. While specific dollar costs are beyond the scope of this paper (because of the detailed description of the parameters and activities which would be needed to understand the costs for some particular evaluation), we know, for example, that a benchmark exercise (comparing just one well-defined workload on two appropriately tuned hardware/software configurations), a procedure in which we have much experience and a large dollar investment, costs several tens of thousands of dollars. Several factors contribute to this problem:
   a. Much machine time is consumed in obtaining valid, reproducible data. Although the technology supporting functional development and testing is evolving toward substantial use of remote terminals, virtual memory and virtual machine systems and other shared resources, performance measurement remains tied to dedicated stand-alone machine time for the calibration and use of performance measurement tools. That is, timing runs and production runs cannot be processed concurrently, in general.
   b. There is a large dollar cost per information unit obtained. Considerable effort is spent regenerating data already available to one analyst simply because its existence is not known or is insufficiently documented to satisfy another analyst's need for control of his variables. Moreover, quantities of raw data are gathered but never reduced or used to their fullest. As Belady and Lehman [4] have expressed it,

"What is missing is not so much data as data analysis. The data is all around us but little is done with it." This, of course, is as much a management problem as a technical one.

c. A third aspect of expense, one common to other advancing disciplines, is the growing effort required to remain current with the state of the art. The body of informal and published literature in the field of performance evaluation is growing rapidly. (Solutions to the problems attendant to this phenomenon were worthy of deliberation at this Workshop.) This is easily illustrated by some of the referenced papers: Lynch [2] provides an overview of current and future positions concerning operating system performance and cites forty references including several bibliographies; Kobayashi [5] in his comprehensive review of the state of the art of system performance evaluation cites forty-nine references, calling attention to recent progress in such areas as workload characterization, statistical analysis of measured results and queuing models, and briefly mentioning simulation models; Parmelee et al. [6] discuss some of the several factors other than main storage overcommitment which affect program performance in virtual storage systems (locality, and the paging algorithm, as examples) in their review of virtual storage concepts, and offer an annotated bibliography containing ninety-six entries, fourteen of which relate specifically to studies of system performance.

2. Performance data has a very short shelf-life. Seldom can a question be answered by obtaining current data and comparing it with previously stored data. Too many variables (whether controlled and documented or not) have changed to permit a useful comparison to be made.
3. A possible third problem area is quoted from a conclusion of a first-line manager of a system performance group: "Explanation of measurement results takes more than 50 percent of the time planned for performance measurements."

These problems (today's costly, transient, and redundant approach to performance evaluation) result from the large, unstructured number of computer system performance factors and the absence of many relational expressions among them. The needs and changes suggested below address these problems.

3. Current Trends and Requirements for Change

We may assess our progress relative to solving today's problems and meeting the requirements of future systems by examining the current trends and then estimating what additional efforts need planning to fully satisfy the long term needs.

3.1. Trends

Among the trends observable today are:

1. The scope of system performance measurements is widening. Some examples:
   a. Configurations are more varied, sometimes now including networks;
   b. Operations and other users' costs constitute an increasing factor in the total installation cost/performance ratio;
   c. Availability, productivity, and value to the end-user are more readily admitted as being legitimate considerations of the performance analyst.

As a result, additional factors and relationships must be investigated and quantified.

2. Systems are being designed to make fuller use of available resources. Prior to the VS systems, a typical system was designed and configured to handle an anticipated peak load. The load (measured by the number of jobs processed per unit of time in batch environments, in terms of message rate for on-line systems) was an object of maximization. Resources were allocated according to a peak demand. For example, main storage was allocated for the duration of a job step based on the maximum instantaneous need during the processing of that step. With the more dynamic allocation of resources in VS systems, all resources are utilized to a greater degree; a balanced use is a system objective, to truly free up and make available the previously unused potential of the systems' resources. Greater overlap (e.g., of CPU and I/O) is an objective. This necessitates some reorientation of performance analysis from independent measurements of resource usage and throughput to the correlation of overlapped resource usage with throughput.
3. With virtual storage systems there is a trend toward greater use of shared resources, exempli-

fied by more programs being reentrant. Since program fetching is less efficient than paging, there is a double benefit: first, a single copy of the program can be shared by more than one task concurrently (reducing program fetching); and second, paging is reduced, since reentrant programs contain more pages which remain unchanged and therefore don't get written back to the paging data set when the page frame in real storage is freed or made available to another task.

3.2. Requirements For Change

Based on the problems facing performance evaluation and the trends seen today, several changes may be appropriate:

1. No more striking need exists for a common vocabulary and technical approach to performance evaluation than the need to communicate between the vendor and the end-user of a data processing system. The performance-related activities of the vendor (designing, developing, selling, installing, and maintaining a system) may be offset in time from those of the user (predicting, configuring, and tuning the system), but common to all of these activities are the same technical complexities.
2. There is the necessity to describe the performance of a system quantitatively. In an earlier paper [7], I noted that regardless of the objectives of evaluating the performance of a data processing system, measured data is invariably required; that generally this data is concerned with the hardware or software resources of the system in its fully operational (available) state; and that any such data falls into one of these four types: utilization (how long a resource is in use), activity (how many times a resource is used), contention (how much demand there was for a resource which could not immediately satisfied), and load (what kind of demand there was for a resource). The major portion of the paper was then devoted to guidelines for selecting the particular data to be measured as a function of the purpose of the evaluation, and selecting which measurement tools to use from the knowledge of what data is needed and what capabilities are offered by the available tools. This progression, from identifying the evaluator and his objective, to specifying the particular data to be measured and the means of measuring it, is illustrated in Figure 3. The paper shows how the current approach of computer system performance evaluation often contrasts with the more disciplined approach required.
3. The collection and display of measured data should be made more dynamic if it is to be effectively used, just as the allocation of resources by the system was made more dynamic to make the system more responsive to the user.
4. Measurement technology needs to become more implementation independent. According to some, the tendency today toward elaborations of existing technology tend to work against innovation and advances of the state of the art.
5. Measurement methodology should proceed inward from the end services performed for the user rather than outward from the details of activities within the system.
6. To accomplish the above, we must learn to characterize the end-user functions being measured independent of implementation, and to identify the common activities. This is a different approach than listing basic or fundamental factors as measures of performance and developing a hierarchy of relational expressions.
7. Means of evaluating tradeoffs among costs, schedules, and technical effects must be improved. For example, without any long-term financial justification for doing so, product development personnel usually persist in the notion that hardware decisions may be made sooner than software ones, since software is inherently more flexible and can be changed faster and more easily.
8. Means of understanding the performance consequences of changes in design, configuration, or workload without the expense of implementing and then measuring those consequences must be improved. As Kimbleton [1] observes after noting several studies which report separate analysis of CPU scheduling algorithms, I/O models, and memory management models, "predictions are normally made in terms of the performance of a subsystem and not the system as a whole."

In the Appendix is provided a more specific statement of needs in this area of improved performance predictability. I suggest that performance factors which satisfy those needs will prove more valuable

than those which continue to account for or explain resource usage. Beyond advancing the technology to the point of satisfying the needs listed there, I suggest these two long term goals regarding terminology and methodology:

1. Performance (and the tools and terms used to measure and describe the factors affecting it) must adapt from a notion of performance as only applying to a "totally up," 100 percent available system to that of performance over a range of availability based on a variable hardware and software configuration. The corresponding functional requirement has been stated by a large user organization [8] as: "The concept of dynamic reconfiguring (including software must) be applied wherever possible * * * It should be possible to alter system parameters, introduce new versions of modules, etc. without re-IPL."

   Other requirements stated in that reference include providing consistency of terminal response time, allowing user control of the overhead tradeoff for performance data collection, and creating a base of data for both tuning and accounting. Performance should be more self-tuning, and require fewer operator decisions.

   The effect on system performance evaluation may well be to introduce yet another type of performance measure; namely, describing as a percentage of the potential offered in theoretically well-functioning, 100 percent available systems the performance actually being delivered in a degraded mode of operation. With this could come the definition of new performance factors involving notions of efficiency, percent of capacity, and of productivity.
2. Composite performance factors providing simplified, user-oriented measures of performance must be developed. The temperature-humidity index, wind-chill factor, and degree-day are examples of such measures which have achieved popular acceptance by the weather forecasters, skiers, and retail fuel distributors, respectively. Other attempts at this include the government's pollution index, the telephone company's message unit, and, of course, the electric company's single billable parameter, the kilowatt-hour.

   These measures have in common an appealing simplicity and practicality which were achieved by forced elimination of precise or elaborate qualification and highly technical terminology.

By contrast, just the mention of the phrase "computer system performance factors" is likely to set off a mental chain encompassing such terms as CPU utilization, instruction mix, address referencing pattern, workload characterization, and many others; I've seen such lists extended to over a hundred items. More than just their professional instead of end-user orientation, these "factors" tend to be measures of resources available versus resources used, rather than in terms of work performed for the economic enterprise which is served by the data processing system.

Of course, there have been attempts at simplified measures. To compare performance across systems for the same workload or across workloads on the same system we have not only the system performance profile (Figure 1) but also several versions of a "standard machine unit," a measure of system power or utilization independent of many details of environmental parameters. Partition-hours is a simple attempt to account for multiprogrammed machine time in terms of yesteryear's batch-operated machine-hours. Chow [9] studied the relationships of such variables as cost, capacity, and access time. Hoja and Zeisel [10] introduced three composite measures to analytically investigate the feasibility of determining the optimal configuration of a terminal-oriented multiprocessing system, given a specific workload. Shedler and Yang [11] considered four variables, dependent on hierarchically more independent variables, to estimate system performance. Haney [12] is attempting to correlate programming manpower and schedules to the structure and complexity of a system using a technique he calls "module connection analysis." Fed probabilities of changes in any one module requiring changes in each other module, he produces quantitative estimates of the effects of module interconnections. Possibly a similar analysis could be made to estimate the effects on system performance of changes to the performance of any component of the system, for as Lynch [2] expresses today's status: "a description of the performance of each part or module of an operating system will, in general, give little clue as to the overall system performance."

An example of an overall approach is reported by Waldbaum and Beilner [13], who developed a total system simulator (hardware, software,

workload, reliability, and operator actions) as a means of improving installation performance in a particular case.

It appears that the range of current investigations spans the entire spectrum of system and workload performance factors in an attempt to establish some fundamental relationships among them.

4. Summary

I have outlined several areas of computer system performance in which our ability to provide more adequate analysis and useful evaluation is limited by current practices wherein additional factors are being identified faster than the relationships among them are being understood. I have suggested that a number of changes are needed in technology, terminology, and methodology for us to advance beyond these limits in the future, and discussed some. Due to the lead time involved in developing and introducing new technology, standards, and methods, we could say that if we are not at least five years ahead of time in our planning and thinking, we may already be behind in our work. The future is already under construction. What may be new is an enlarged awareness and a sense of urgency, a growing conviction among data processing users and vendors alike that we must become more action-oriented in our planning and do those things soon that need doing.

5. References

[1] Kimbleton, Stephen R., "The Role of Computer System Models in Performance Evaluation," Comm. ACM, 15, 7 (July 1972), pp. 586–590.

[2] Lynch, W. C., "Operating System Performance," Comm. ACM, 15, 7 (July 1972), pp. 579–585.

[3] Green, Joseph H., "Coding Techniques for Virtual Memory," IBM Technical Report TR 00.2332, New York Development Center (June 20, 1972).

[4] Belady, Laszlo A., and Lehman, Meir M., "A Systems Viewpoint of Progrimming Projects," Imperial College, London Research Report 72–31 (Aug. 1972).

[5] Kobayashi, Hisashi, "Some Recent Progress in Analytic Studies of System Performance," IBM Research Report RC 3990, T. J. Watson Research Center (Aug. 17, 1972).

[6] Parmelee, R. P. et al., "Virtual Storage and Virtual Machine Concepts," IBM Systems Journal, 11, 2 (1972, pp. 99–130.

[7] Morrison, Robert L., "An Approach to Performance Measurement," IBM Technical Report TR 00.2272, Poughkeepsie Development Laboratory (Feb. 15, 1972).

[8] Lasky, M.D. et al., "TSO/OS Project 'White Paper' on Future Systems," SHARE, Inc. (Aug. 1972).

[9] Chow, C. K., "On Optimization of Memory Hierarchies," IBM Research Report RC 4015, T. J. Watson Research Center (Sept. 5, 1972).

[10] Hoja, H., and Zeisel, G., "Measures for the Quality of a Multi-processor System," IBM Technical Report TR 25.131, Vienna Laboratory (Sept. 2, 1972).

[11] Shedler, G. S., and Yang, S. C., "Simultation of a Model of Paging System Performance," IBM Systems Journal, 10, 2 (1971), pp. 113–128.

[12] Haney, Frederick M., "Module Connection Analysis—A Tool for Scheduling Software Debugging Activities," Proc. 1972 AFIPS FJCC, AFIPS Press, Vol. 41, pp. 173–179.

[13] Waldbaum, Gerald, and Beilner, Heinz, "SOUL: A Simulation of OS Under LASP," IBM Research Report RC 3810, T. J. Watson Research Center (Mar. 30, 1972).

APPENDIX

Performance Predictability Needs

1. Ability to translate system performance statements (e.g., objectives and specifications) into component performance statements, and vice versa

2. Ability to determine how changes to machine and program architecture will affect system performance

3. Ability to determine performance effects and costs of shifting function to a different component (e.g., software to hardware)

4. Ability to quantify on an incremental basis the performance effects of adding, deleting, redefining system functions

5. Ability to determine from a component's performance in an existing system what it will be in a new system.

6. Ability to identify for each component the other components on which the performance of the given component depends, and to quantify these relationships

7. Ability to specify values of design parameters (e.g., #buffers, #phases, #fixed pages, . . .) from objectives

8. Ability of designers and implementers to estimate resource (hardware and programming) usage of various design possibilities

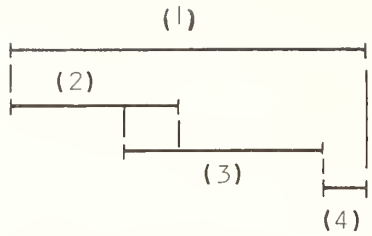
9. Ability to get feedback during implementation about actual values of design parameters and resources usage and a procedure to relate this to objectives and effect necessary reconciliation

10. Ability to reconstruct after development the rationale (performance consequences) supporting major design decisions during development

11. Ability to estimate the potential performance of a specified combination of hardware, software, and workload

12. Ability to determine how much of the potential performance of a given system is being achieved by a given installation using that system

13. Ability to assess the performance of a given system relative to that of competing or alternative systems



(1) Total elapsed time
(2) CPU busy time
(3) Any channel busy time
(4) System wait time

Figure 1. The System Performance Profile

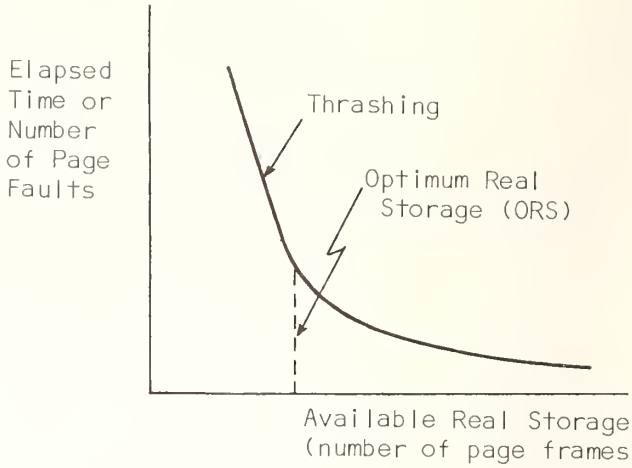


Figure 2. Dependency on Real Storage Availability in Virtual Storage Systems

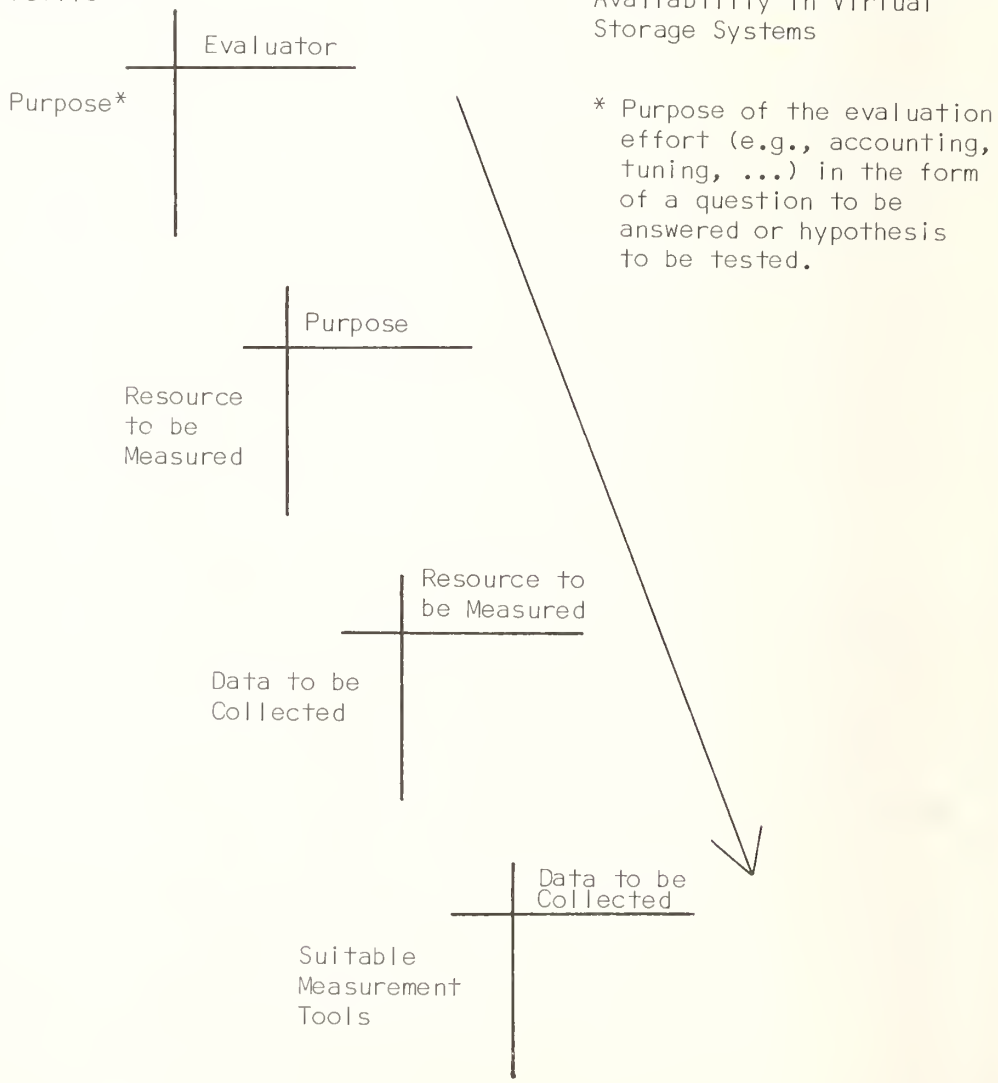


Figure 3. Focusing on the Relevant

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

End-User Subsystem Performance

Philip J. Kiviat

Federal Computer Performance Evaluation and Simulation Center, Washington, D.C. 20330

Subsystem end-users should be concerned primarily with measures of system effectiveness that are cost or value based. Only through these measures can they relate the operations of their subsystem to the goals of the larger system. Individual subsystem effectivenes measures should be related through a total systems effectiveness model to permit tradeoff and marginal allocation decisions to be made.

Subsystem end-users are usually not concerned with measures of system efficiency, which are the traditional computer performance measurements, but they are responsible for seeing that their effectiveness is achieved at minimum cost, which is determined and achieved by analysis of computer performance measurement data. Subsystem end-users therefore should see that their operating units receive system efficiency measurement data and that they understand how resource efficiency is related to system effectiveness.

Key words: Computer performance evaluation; efficiency measurement data; measures of computer performance; system effectiveness.

1. Introduction

Any discussion of "End-User Subsystem Performance" cannot proceed without a clear definition of the terms: User, End-User, System, Subsystem and Performance. Without such definitions any discussion will tend to be over generalized and not amenable to useful summary and conclusion.

The following definitions will govern the comments made in this paper:

User: An entity that receives service from a system or subsystem.

Examples: a. A programmer using a computer facility for production or development work.

b. A teller using an inquiry terminal for account status interrogation.

c. A real-time experiment using a computer for data collection, analysis, and parameter control.

End-User: The person at the end of a user-subsystem chain who receives the ultimate system service.

Examples: a. The marketing analyst who gets his daily sales reports from an applications programming group.

b. The financial vice-president or controller who manages his company's credit system through reports generated by his retail point-of-sale computer terminal system.

System: A collection of people and equipment organized to provide a service (or services) to one or more end-users.

Examples: a. A reservations system designed to match people and non-sharable resources.

b. A tracking network designed to monitor the progress of a satellite as it orbits the earth.

c. A time-sharing service that provides computational facilities to remote users on demand.

Subsystem: A system that provides an identifiable service to a user (either a person or a subsystem) and is either an operating part of a larger system or shares some resources with another system.

Examples: a. The reservations subsystem of an airline's total computer operation.

b. The time-sharing component of a full-service computer service organization.

Performance: Some measurable quantity that relates the function performed by a system or subsystem to either the resources required to perform the function or the rate, quality or reliability with which the function is provided.

Examples: a. Cost per teller inquiry.

b. Response time to a teller inquiry.

c. Time to develop a new applications program.

d. Cost to provide a unit of computing power to a customer.

Ideally, the performance of a system should be judged by measurements of the performance of the services it provides to its end-user. However, the need for control and decentralized management, and the lack of integrated system performance models forces the use of end-user subsystem performance measures as surrogates for global performance measures.

Consider the following: A large retail chain installs point-of-sale terminals at all its sales points in all its stores. The terminals are connected to a central computer that also does the chain's accounting, inventory control, payroll, etc.

The end-user of the computer system, the company president, can see whether net profits have increased after installation of the POS terminals, but he can't know whether profits can be improved or how various components of the POS system contribute to profits, nor can he control the operations and evaluation of the system with balance sheet and P&L statistics. For these purposes: management, control, design, determination of efficiency as opposed to effectiveness, subsystem performance measures must be used. And since decentralized management decisions must be made, end-user subsystem performance measures must be used within a total systems performance framework to guard against harmful suboptimizations.

To be more precise, end-user subsystem performance measures of effectiveness must be developed that act as surrogates for end-user system performance and end-user subsystem performance measures of efficiency must be developed that act as hallmarks for local control.

Consider the POS subsystem. A set of performance measures must be chosen that capture the impact of the subsystem or system effectiveness rather than display some visible subsystem product, i.e. collectable sales dollars versus reduction in the rate of bad debts by instant credit checking. What good is a zero bad debt rate if most customers get so fed up with waiting in line at the terminal that they stop shopping at the store?

Yet to design an efficient subsystem, goals must be specified for component designers that are necessarily removed from or have a complex and indirect relationship to system effectiveness.

2. Categories of Performance Measures

The ultimate measure of any system's value is its effectiveness—is it doing its job? Effectiveness can be measured in many ways: in terms of time taken to respond to a demand for service, in terms of quality of service rendered, in terms of the rate at which requests for service can be honored, etc. The ultimate unit that relates incommensurate effectiveness measures to each other is the dollar: what is a unit of response worth, what is it worth to discriminate at a 95 percent rather than 85 percent level. Performance measures are of little value unless they can be reduced to a common denominator. For example, the number of reservations or jobs processed per employee per hour is of no utility without the value of the jobs that are processed. We all know full well the ability of people to "beat the system", e.g. a clerk will select people with easy jobs to process although the return per person is low, to increase his processing rate, and leave the more demanding, highly profitable customers to fret, fume and depart.

Measures of performance should be chosen that either singly, or in combination with other measures yield dollar values. Rates alone are not adequate measures unless it is truly known that

there is no inverse relationship, either present or possible, between rates and dollars.

Efficiency in the employment of resources is also viewed as a performance goal, as indeed it is. We can call a system efficiently designed if it employs the fewest resources to achieve a stated effectiveness goal. Efficiency is not to be confused with utilization, for while high utilization probably follows from high efficiency, the reverse is not true. High utilization is often a symptom of low efficiency when viewed from a perspective of effectiveness. One reason that high utilization and high efficiency are not necessarily related is that many systems require that a reserve capacity be present so that they can respond quickly to random events. In these cases, high utilization equals high efficiency if a correction is made for reserve requirements.

Efficiency is only meaningful in terms of effectiveness, and measurements made of how resources are used cannot be interpreted out of context.

3. System Design

Over the life of any system there is a common cycle of events: design, production, operation, measurement, evaluation, redesign, production, operation, * * * . During the design of a system performance goals are set, and detailed work assignments made based on the performance goals and the technology available, e.g. before telecommunications it would have been unrealistic to expect an airline reservation clerk to process as many reservations using telephone communications as that same clerk can process today. The subsystem performance goals should be set from a knowledge of their effect on total system performance, i.e., from a model of the system.

Once subsystem performance goals have been set, system designers attempt to produce hardware and software configurations that meet these goals efficiently. For example, given that a reservations system is to be designed to process N reservations per minute, or to process $N \times 10^M$ events per second, then an efficient design will accomplish this with a minimal expenditure of system resources.

Every system should be designed to measure and report both its effectiveness and its efficiency. Effectiveness data should be transmitted from each subsystem to the system end-user who is in a position to correlate total system performance with individual subsystem performance measures.

Efficiency data should be transmitted only to the level responsible for providing the service required to achieve the subsystem goal, e.g. channel utilization, terminal waiting time and CPU waiting for I/O percentage are reported no higher than the computer center supplying the subsystem computer service. This center can use the efficiency (utilization) measurement data to reconfigure or tune the computer system to provide the required service at least cost.

A design principle must be that system and subsystem performance measurements are designed into a system, and that these measurements form a set complete enough to detect proper and improper system performance. For example, it is probably always true that high levels of utilization coupled with low measures of effectiveness mean that a system bottleneck or problem exists at the resource having high utilization; e.g., having a high CPU busy utilization and a low transaction processing rate may mean that a faster CPU or memory is needed, or that a recent data base reorganization increased rather than decreased table search time.

Since hardware forms only one part of a system—software and people being two other important parts—system measurements must go beyond mere counters of hardware utilization. As it is impossible to tell what utilization statistics mean without knowing the conditions under which they were created. measurements must be taken of the pattern of system inputs. A system is designed to process transactions that occur at a given rate and pattern, if the rate and pattern change the ability of the system to process the transactions efficiently changes, and a different level of utilization may be required to meet the desired level of performance. For example, if a new receptionist in a doctor's office submits her patient transactions individually throughout the day rather than batching them at the end of the day, then the number of "doctor information table" accesses will be higher and CPU time used in reading data will be higher. As the doctor probably pays a fixed price per transaction, the increased use of system resources reduces the system's profit potential. The effectiveness measure "transactions processed per day" may be high, indicating good system performance, but so will the "cost per transaction" measure, indicating poor system performance. Without a measurement displaying system workload characteristics it would be most difficult to place the blame on the doctor's office rather than the computer system, and take appropriate corrective action.

4. System Measurement

Unless a system is designed to be measured, it may be impossible to measure it.

System-effectiveness data have two principal dimensions: time and cost. Common time measures are job units processed per unit of time (throughput) and units of time required to respond to a stimulus (response time). It is generally not sufficient to report only aggregate time measurement data, such as the average throughput or average turnaround time. Histograms of time measurement data collected over variably set time intervals are required to infer such things as the probability of a response within some arbitrary number of time units or the probability of achieving a throughput during a congested period of at least some specified rate. The width of the histogram class interval has to be variable to achieve efficient processing and produce analyzed data in the most reduced form.

Cost information can be obtained if system resource usage is measured and applied against a costing algorithm. Quite often such an algorithm does not exist at a subsystem level, and the data that would be used as input to such an algorithm is used instead, e.g. CPU seconds instead of the cost of a CPU second. The difficulty with this approach is that it does not allow a trade-off between different resources to accomplish the same task to be made in the format of a single measure of effectiveness. It is sufficient for cost information to be reported at an aggregate level only, i.e. from resource utilization figures over an interval rather than from analysis of individually computed job cost figures, as management reactions are to average cost levels.

Another measure that is important in determining system effectiveness is value. At times, as in a time-sharing service, time measurements are equated with value. Here a fast response is taken to be more valuable than a slow one in that it leads to higher programmer productivity. (This may not always be true.) In other systems, the value of a task is intrinsic to the task and cannot be observed by a purely passive observer. This is the case in a POS system where the value of a transaction is recorded in its data. Value measurements are important in situations where people influence the flow of data and/or transactions into a system, and their behavior is not observable but is measurable.

System efficiency data is normally obtainable even if a system has not been designed with measurement in mind. Hardware and software monitors can measure the utilization of devices, as well as their interaction in defined states. It is often more difficult to define the measure of efficiency than to measure it. For example, one can measure the utilization of the PPU's on a CDC 6000 system. But is it the percent utilization of the PPU's or the percent of time the CPU was held up because a PPU was not available that is the appropriate measure of how well the PPU's are used.

Utilization data should be reported in time series form, over quanta determined by the unit activity level of the resource, for resources that are shared. As has been pointed out, a two-thirds utilization can come about by 200 units of the time at 100 percent, followed by 100 units of time at 0 percent, or by patterns of 20 units at 100 percent followed by 10 units at 0 percent. These patterns can yield vastly different conclusions depending on the nature of the system workload.

5. Summary

Subsystem end-users should be concerned primarily with measures of system effectiveness that are cost or value based. Only through these measures can they relate the operations of their subsystem to the goals of the larger system. Individual subsystem effectiveness measures should be related through a total systems effectiveness model to permit tradeoff and marginal allocation decisions to be made.

Subsystem end-users are usually not concerned with measures of system efficiency, which are the traditional computer performance measurements, but they are responsible for seeing that their effectiveness is achieved at minimum cost, which is determined and achieved by analysis of computer performance measurement data. Subsystem end-users therefore should see that their operating units receive system efficiency measurement data and that they understand how resource efficiency is related to system effectiveness.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

End-User System Performance

Norman R. Nielsen

Stanford University and Wellsco Data Corporation*, Menlo Park, California 94025

The end user's view of a computer system's performance is generally quite different from that of the computer professional or of the service provider. He is unconcerned about such traditional system performance measures as CPU utilization, channel balance, memory fragmentation, and I/O queues. He is concerned only with the indirect effects of these measures as manifested in the cost he incurs or in the performance he receives. Factors reflected in these measures encompass items in the areas of accounting cost, control, system service, reliability, user interface, output, programming, and user (rather than system) performance.

In addition to the usual perceptual differences that exist between server and user, there are also significant disparities in the items taken to define performance, in the measures used to reflect that performance, and in the criteria employed to evaluate the measured performance. The paper explores some of these differences as well as discussing certain aspects of system performance which are of particular concern to the end users of computer systems.

Key words: Computer performance evaluation; computer resource allocation; computer service parameters; computer system performance; cost/effectiveness; cost/performance; performance evaluation; user control of computing; users' performance evaluation; users' performance measures.

1. Introduction

The end user's view of a computer system's performance is generally quite different from that of the computer professional or of the service provider. In addition to the usual perceptual differences that exist between server and user, there are also significant disparities in the items taken to define performance, in the measures used to reflect that performance, and in the criteria employed to evaluate the measured performance. It is the purpose of this paper to explore some of these differences as well as to discuss certain aspects of system performance which are of particular concern to the end users of computer systems.

2. The End Users

In the final analysis the ultimate end users of computer services are the presidents or chief executives of those organizations using such services. However, it is generally more illuminating to consider some of the intermediate end users that fall in the spectrum between the aforementioned ultimate end users and the direct users of computer system services. Under this approach the end users in commercial data processing applications would be considered to be the departmental and divisional managers, not the data control clerks. In an educational setting faculty members and department chairmen would be taken as the end users rather than students or programming assistants. In a programming or system development environment it would be the project leaders and development managers, not the programmers and analysts, that would be the end users. Similarly, in on-line reservations, banking, or ticketing operations the end users would be reservations managers and vice presidents for sales rather than agents or tellers dealing with the public.

Thus the term end users is used in this paper to refer to those individuals (at all levels) who have

* Now located at Stanford Research Institute, Menlo Park, Calif. 94025

responsibility for the work that is being processed by the computer system and for the personnel working with that system (providing it with inputs, working with its outputs). However, these individuals are not direct users of the system themselves. Furthermore, it should be recognized from this definition that these users are not concerned with the technical characteristics of the computer's performance except insofar as it impacts the performance of their department or operation. Even when they are so impacted, their concern is likely to focus upon that impact rather than upon the underlying reasons at the computer facility or upon the resulting impact felt by other users of the system.

3. The Basic Performance Measure

In principle the end user has but a single measure of computer system performance. This measure is simply the "cost" to do the "job." The term "job" is used rather broadly here, so that it refers not only to the computer job but also to the entire user system which is serving the function for which the end user is responsible. Thus, a computer system is only a part of the larger system that must be considered. Clerical staff, mechanical equipment, and even transportation services may constitute a part of the system performing the job.

The term "cost" is also used broadly. Clearly it should be stated in units that are appropriate for the end user and his environment. Thus, cost might be stated in terms of dollars/year, dollars/reservation, dollars/student, or other monetary unit/unit of work. The cost should be calculated so as to cover the "true cost" of providing the service or performing the function for which the end user is responsible. Clearly this will include an allocated or billed charge for the computer or computer resources used, for the salary and space costs of computer support personnel, for peripheral and ancillary equipment, etc. Total computer costs are generally relatively straightforward to calculate for an existing operation (and to estimate for a projected operation). (The allocation of these costs, though, as charges to end users is much more complex and raises a number of other issues that are beyond the scope of this paper.) To these allocated computer charges the end user must add his own costs for staff, space, other equipment, etc.

However, there is still another aspect that must be considered, and this relates to performance costs. System response or throughput may influence staffing levels and customer satisfaction. System capability may affect transportation costs. System reliability may also impact the operating unit's costs, including those attributable to missed deadline penalties, customer dissatisfaction, opportunity losses from missed sales, overtime and idle staff time, staff morale, commitments for standby facilities, etc. These performance related costs can be significant, and only by reflecting them can a true cost picture be obtained.

The above described "cost/job" measure will suffice to make definitive comparisons between different computer systems, even if one should desire to compare systems with different speeds, architectures, capacities, or software. This measure is still appropriate in those cases where the computer systems being compared are not direct substitutes (e.g. employing different mixtures of people, mechanical equipment, and computers in the operation). It is important to note that, even though the systems being compared might not provide equivalent service (or meet the desired specifications equally well), the performance cost figures would correct for these differences. Thus, a single all encompassing number can be used as a meaningful end user performance measure.

4. Performance Measure in Practice

Although the above described cost/job measure depicts in principle how an end user should measure the performance of a computer system, it can not really be advocated as a practical procedure. Many of the considerations are difficult to measure or to judge, others are simply too "fuzzy." Thus, for example, how does one measure the additional staff that might be necessitated due to the greater software complexity of one system versus another? Or how does one estimate the resources that will be lost as the result of a tedious job control language? On the non-computer side there are similar difficulties in measuring the cost of a dissatisfied customer, of sagging staff morale, etc. These measurement difficulties are frequently compounded by an end user's inability (or unwillingness) to make explicit trade-offs. Thus, it may be very difficult to obtain a user's indifference curve between lower probabilities of service disruption on the one hand and higher cost with greater customer

satisfaction and greater staff efficiency on the other. As a result it is very difficult to assign costs to some of the performance impacts; even if they could be measured or estimated accurately.

Consequently, in some of these fuzzy areas managers are prone to make these trade-offs implicitly rather than explicitly. In other words, trade-offs are made only on a case-by-case basis and usually only on a "better or worse" basis rather than on a relative scale. Thus, situation A (consisting of accounting costs, service levels, morale, etc.) would be compared with situation B, and one would be judged the better. The various criteria would be compared and aggregated implicitly, without ever explicitly attaching weights to the various criteria or attaching costs (benefits) to the resulting measures.

In such an environment resort is often made to cost comparisons for some given level of general performance. The differences in performance (e.g. system reliability, ease of use) are all thrown together and evaluated implicitly as a package against the ordinary or accounting type cost for the system. Thus, the measure of system performance from the point of view of the end user becomes one of cost for a given level of performance (or cost/performance for short) rather than the overall cost of the performance.

These measurement difficulties clearly point to the need for further research on both the proper measures of some of these end user aspects of system performance as well as the means to make the measurements. On the computer side there is a need to measure staff productivity in connection with the type of service provided, the quality of that service, special features offered, etc. The impact of time-sharing service upon programmer productivity is an illustrative case. Most of the other problem areas touched upon above are not so much computer problems as they are application problems. The cost of customer dissatisfaction, for example, is very dependent upon the operating environment and is unrelated to the computer system. Even the relationship between a computer system's performance and the resulting level of customer satisfaction is dependent upon the application and the environment. Thus, many of the more pressing questions are not potential candidates for research by computer scientists. They are, however, definitely in need of study by persons in other fields and disciplines.

Given the use of cost/performance as the end user's measure of system performance, it is appropriate to consider in further detail some of the factors which constitute this measure.

1—Cost: This factor might be termed the accounting cost in order to distinguish it from the overall cost that was described previously. Accounting cost encompasses the direct costs for using the computer system as well as some of the more obvious indirect costs. Direct costs cover the allocated costs of the computer operation (e.g. equipment, space, operating staff). The end user may incur these directly (the computer being part of his area of responsibility) or by allocation (computer service being "purchased" from another operating unit). Indirect costs might include contract or missed deadline penalties (e.g. lost float in a banking application), certain staffing or overtime costs attributable to computer system performance, reruns necessitated by errors, etc. The accounting cost can best be characterized as the "obvious costs." Depending upon the accounting system used in the organization, the costs so developed may or may not be an accurate reflection of the real impact or actual costs in these areas. However, these are the cost figures which are generally used in performance evaluation.

2—System Service: Basically this factor covers the turnaround or response received by the direct users of the computer system. The service factor is of particular importance to the end user due to the impact of customer service levels upon staff productivity and upon the service levels that can in turn be provided to the customer. Thus, the end user is interested in the system's mean response time as well as the distribution of the response times received. He is also interested in the ability of other users to degrade his service and in the control he can exercise over the service level to be received by his staff (e.g. use of priorities, rescheduling of workloads). The availability of effective control mechanisms is particularly important in environments characterized by deadlines.

In order to provide the best possible service the computer facility staff will be involved in a host of activities, including the refinement of scheduling algorithms and memory allocation procedures, the balancing of channel activity, the configuring of systems, etc. A variety of system measurement activity will generally be connected with these activities. However, all of this should be invisible to the end user. He cares little for the efficiency of the scheduling algorithm, for example, so long as it gives his staff either (a) the desired service levels, or (b) the

parameter options that will permit them to obtain appropriate service levels as it becomes necessary.

3—Reliability: Although the end user is not concerned about the reliability of any given piece of equipment (as computer facility personnel might be), he is very concerned with the reliability of the end product (service) provided to his staff. Thus, the frequency and duration of service outages from any cause (including inadequate backup and recovery procedures) is of interest as is the extent, frequency, and duration of degraded service due to hardware failure, software failure, facility failure to regulate demand, etc. Attaching a cost to such unreliability and including it in the first category above would be very desirable, but this is generally not realistic at the present time.

The impact of service reliability is very dependent upon the application context. At one end of the spectrum there is almost no effect (e.g. a quarterly update that can be processed anytime during a week's interval). Further along the spectrum are the cases of general delays. Service outages merely cause everything to slip forward in time (e.g. the result in most program development situations). There are some immediate costs of lost time (e.g. staff salaries and frustration) as well as the longer run slippage cost in project completion (e.g. delay in receiving anticipated savings). At the far end of the spectrum are the cases where there is serious and immediate impact (e.g. situations characterized by service deadlines or contractual output requirements). The end user in this type of situation needs not only a measure of the service reliability but also some form of control over the service level and the attention the computer facility gives to reliability (e.g. supplemental service arrangements). This matter of control is addressed at greater length in the section on user controls below.

4—User Interface: This factor covers the ease with which the direct users can use the system. It reflects the "personal overhead" involved in specifying priorities and setting up control cards or otherwise preparing for a run, the time and difficulty in making contact with the system from a terminal, the amount of training required for new personnel, the frequency with which system changes necessitate a change in staff procedures, etc. The cost effects of these items tend to be non-quantifiable, but there are generally good qualitative feelings for these variables.

5—Output: This factor encompasses a variety of items relative to the output from a computing system. An obvious item concerns the quality of printed output, including proper form usage, form alignment, type cleanliness and alignment, ribbon quality, etc. Another item concerns the identity of the delivered output (e.g. yours not someone else's), the order and completeness of the outputs, etc. There should also be a concern with the correctness of the outputs in the sense that all of the proper job steps were run (and run only once), that no input data was accidentally loaded twice by an operator, etc. Despite the high visibility of these areas to the user, there is generally little if any regular performance measurement activity along these lines.

6—Programming: Although not a computer service per se, this factor is so closely related to the computer system performance seen by the end user that it should also be considered. Clearly there are the normal programming costs for staff and machine time. However, there are a number of other important considerations which a computer system can influence, including the speed with which programs can be developed, the efficiency with which the programs will operate, the ease with which the programs can be used, the degree to which these programs can serve the needs of the end user, the ease with which bugs can be tracked down and eliminated, the ease of maintaining and modifying existing application programs, and the special features and facilities provided by the system. Again, there are often good qualitative feelings for these items, but there is generally little in the way of quantitative performance measurement.

7—User Performance: A final factor, and one which is of interest to both the end user and the computer facility, covers the performance of the direct users. This would include such items as late inputs for scheduled runs, improper inputs (e.g. providing the wrong tape reel), erroneous data and poor quality media as well as terminal input speeds and error rates. Real-time and conversational systems sometimes measure some of these aspects of user behavior; most of these items are not monitored in batch systems.

In reviewing the content of the above described seven factors that constitute cost/performance, one can readily see the extent to which many of the traditional measures of system performance are absent. The end user is not concerned with CPU utilization or similar measures so long as the service which he receives and the price which he pays for the service are "acceptable." In other words, the end user judges a computer facility on the cost/performance of the

service (product) it delivers, not on the manner in which it produces that service.

5. User Controls

The end user is primarily concerned about the non-computer aspects of his operation. The computer portion of the activity is "contracted out"—at least logically if not physically. This is why the performance considerations discussed thus far all focus upon system performance as viewed at the interface between the service provider and the service user. This also highlights the need to consider in somewhat more detail the means by which the user and the computer facility can relate to each other across this interface. For the purposes of discussion a user's processing might be divided into four categories.

1—Scheduled Batch: Work in this category is often characterized by a reasonable predictability of its computer resource requirements and of the time at which those resources are required (e.g. an hour everyday at 7:00 pm). Since work in this category is of a recurring nature, the user and the facility can invest the time to negotiate a cost-service schedule. Ideally, the user would specify the input time, the desired output time, and the resources required (probably derived from system accounting data on past runs). The facility would then offer a rate for this service based upon the job's resource and turnaround requirements and upon the facility's general system load. Presumably this would lead to negotiations concerning price adjustments in response to changing resource or turnaround requirements.

Although such an interchange would be very beneficial, it rarely takes place in as explicit a fashion. Rather, somewhat fixed resource rates are established, and the user and the facility then haggle over the schedule or service to be provided (with politics perhaps becoming involved). Another problem concerns the significant difference in the units which the two parties employ in this process. The computer facility thinks in terms of $/CPU minute, $/track of file space, etc. The end user thinks instead in terms of $/order processed, $/account record stored, etc. If the computer facility adjusts its thinking (rates quoted in user units), then it must absorb any differences between charged usage and actual resource usage. If the computer facility does not adjust to the user (rates quoted in resource units), the user will translate actual charges back to user units and will see a high day-to-day variance in these effective rates.

Finally there is the problem of delivery on the schedule (fulfillment of commitments). The fact that the facility suffered a hardware failure, or that a software bug slowed down production, or that another department's late inputs threw off the schedule is interesting but not too meaningful to the end user. Such "excuses" are of little consolation to a user faced with his own output schedules, contractual or otherwise. Thus, he needs to be able to exert some influence or control over the operation of the facility in order to protect his own deadline commitments. Aside from "screaming" or exerting "political muscle", he is often helpless. A possible vehicle for overcoming this problem is the specification by the user of a penalty or discount function for each scheduled job or work unit.

Such a function would serve two purposes. First, it would provide a late work indemnity to the user in the form of an outright payoff or a discount in his processing cost. Not only would this insure attention by the computer facility, but it would also provide some measure of consolation for the user. Second, such a function would provide a variety of guidelines to the computer facility, such as the relative importance of a job (for on-the-fly scheduling after a failure), and the general capacity as well as the backup capability required. It is generally obvious that a facility should install only enough equipment to meet X percent of its deadlines in the course of a month. However, the determination of the value of X without information similar to that provided by the penalty functions is very difficult.

2—Interactive Production: Work in this category is often characterized by a predictable resource demand per interaction and by a user concern for service availability, response, and cost. Service unavailability can have serious consequences for the end user's operation in this type of environment. Hence, as for scheduled batch, there is a need for user protection and facility service guidelines. In this case, though, processing may be scheduled in terms of response times for given transaction levels during given time periods (e.g. 3-second responses for demand rates up to 45 per minute between 9 a.m. and 11 a.m.). Presumably a service facility would wish to protect itself in such a situation either by refusing to accept higher demand rates during the specified time period (so that it could assure itself of meeting other commit-

ments) or by accepting a temporarily higher demand rate in return for a temporary waiver of the response requirement. Operating in this type of an environment may necessitate additional performance measures—not only for after-the-fact performance evaluation, but also for control purposes in real-time. Oftentimes the measures that would be useful for this purpose are not among the standard ones.

3—Interactive Development: Work in this category encompasses unscheduled and non-production types of processing. It is often characterized by a relatively unpredictable resource demand per interaction and by a user concern for service availability and response. Not only is there less control capability as a result, but there is generally less information available about the service actually being provided. Often resort must be made to measures of good response (i.e. less than Y seconds) for X percent or more of the requests, with X being the figure of merit. Such a measure does not, however, reflect the differences in the resources actually required during each iteration. Thus, it is a fairly gross indication of performance. The use of such a measure also poses a problem for the user. Aside from programmer unhappiness, very little is really known about the effects of delayed response upon productivity. Predictable response is better than unpredictable, and response can vary with perceived processing required without effect. There does seem to be some optimal point up to which greater delay is advantageous and beyond which human performance deteriorates. However, knowledge in this area is so far very sketchy. Even the more straightforward relationship between programmer productivity and type of computer service (e.g. batch processing, time-sharing) is not really known. Thus, much more remains to be done with respect to the development of appropriate measurement tools and techniques for this category of work.

4—Unscheduled Batch: Work in this category is often characterized by a wide variety of resource requirements that are desired "on demand." The user is often concerned primarily about turnaround and cost. In this type of environment it is not feasible for the computing facility to negotiate service requirements with each user for each job and then to schedule those jobs, for volume considerations are generally overwhelming. Thus, the brunt of the load leveling or scheduling uncertainty must be passed along to the user. Although there are many ways in which this can be accomplished, the following three techniques are indicative of the range of possible approaches.

a) First-come—first-served: This procedure may be applied strictly to all work or only to unscheduled work on a resource available basis following the scheduled workload. In either case a fixed rate is established for computing, and the service received by a user is a variable. This results in a misallocation of resources unless all jobs are considered of equal importance. Further, a great deal of user effort will often be devoted to pleading for exceptional treatment, to developing sub rosa procedures to get work in at the front of the queue, etc.

b) Priorities: This procedure permits the user to select the priority he desires for his processing. A fixed but different rate is attached to each priority. Thus, there is a variable cost for computing depending upon the computing load and hence upon the priority that the user must exercise to obtain the anticipated service level desired. (Alternatively, the user can always select a given priority and thereby face a constant rate; however, his service will vary with system demands.)

c) Variable Rates: This procedure permits the user to select the desired service level. The rate charged for that service will vary, however, depending upon system demand. This approach has the advantage of eliminating the need for a user to estimate the priority required to obtain a particular service level at a given point in time.

The above approaches progressively remove more and more of the uncertainty from the user with respect to the service to be received, but they achieve this at the cost of more and more uncertainty about the price to be paid for that service. (Given that computing is a limited resource, it is not possible for a user with unscheduled work to specify both his cost and his service.) Although the end user is concerned about the service received and the cost incurred by his staff, the "other costs" incurred in working with the system are also important. Thus, the impact of the service actually received, of the uncertainty in service levels and rates, and of the time and effort required to interface with the resource allocation mechanism must be evaluated.

Despite the above comments the selection of a scheduling procedure is not the sole province of the user, for the computer facility is also affected. Thus, while procedure (a) impacts the user the most with

regard to service variability, it is the cheapest to implement and operate. Procedure (c) frees the user from system load estimation but only at the cost of much greater measurement and control work by the computer facility.

As for the interactive development work category, the greater variability and unpredictability of unscheduled batch makes performance more difficult to evaluate. Again, too, there is the problem of the end user being able to evaluate the worth of a given service level to his department. When alternative scheduling and control systems are included in the consideration, there is the further problem of evaluating the fuzzy cost-benefits that result.

6. Summary

The end user is unconcerned about many traditional aspects of computer system performance. Thus, such system performance measures as CPU utilization, channel balance, memory fragmentation, and I/O queues are of little interest or value. The end user is concerned only with the indirect effects of these measures as manifested in the cost he incurs or in the performance he receives. In this connection there are a variety of user oriented performance measures that are frequently employed in practice. These measures are of value not only to users but to computer facilities as well. The factors reflected in these measures encompass items in the areas of cost, system service, reliability, user interface, output, programming, and user performance.

Most of the items covered by these factors are primarily user oriented rather than operationally oriented (although characteristics of the latter are involved). Measures such as turnaround, ease of use of resource allocation procedures, contract deviations, and quality of delivered output are illustrative. However, user measures can lead to the need for additional measures on the part of the computing facility in order that it may perform in a user oriented environment. For example, consider the variety of data required if a facility is to develop rates for guaranteed levels of service.

Thus, although end user performance measurement does not employ many of the traditional measures of computer system performance, it does require a variety of additional measures. Not all of these measures are yet well defined, and still others have somewhat vague interpretations. Clearly there is much work to be done in this area, particularly on user impact or application dependent effects (although the "computer measurement" aspects are not completely in hand either).

Evaluating computer system performance from the viewpoint of the end user provides a more meaningful guide to these the ultimate users (who are afterall the real justification for the computer system). Further, such measures can provide perspective and direction for the computing facility. They do add an additional level of complexity to the measurement process, and they can lead to some new areas of concern. They also have their costs. This, however, is true for all forms of measurement, and the cost-benefit of each desired measure must be evaluated before it is added to the measurement and analysis load. In situations where there is a clear payoff, the measure should be implemented promptly. In other cases a need for further research may be indicated in order to reduce the cost of making the measurement, to develop alternative or surrogate measures, or to find a better or additional use for the measure so as to increase its value.

C. COMPLEMENTARY CONSIDERATIONS

Several pursuits are closely linked to performance analysis because they all use common tools for a common purpose—to determine the types and degree of interactions in computers. In addition, activities in these related fields can strongly influence the performance attained on a computer system. Some of these fields are discussed in the papers below:

- Computer Design, by Wilner;
- Software Validation, by Rubey;
- Data Security and Privacy, by Chastain;
- System Reliability, by Hughes.

Their objective is to consider both the applicability of performance analysis approaches in the related fields and the degree of influence of the related fields on computer systems performance analysis. Each contains some discussion of the following topics:

1. Experience in the related fields that illustrate the interactions between those fields and performance analysis.
2. Problems encountered in the related fields due to inadequate or inappropriate measures/measurement tools.
3. Specifications for performance analysis work that would aid efforts in the related fields.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION:
REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Complementary Pursuits—Computer Design

W. T. Wilner

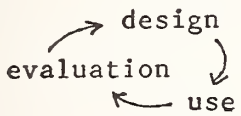
Burroughs Corporation, Goleta, Calif. 93017

The relationship between computer design and performance analysis is argumentatively claimed to be an information-producing symbiosis. Performance analysis can add precision to the conceptual models which designers use to generate new systems. Most of the major aspects of good models, however, are unquantifiable. Computer design can help or hinder performance analysis, mainly by adding or omitting those few components which allow hardware monitors to recognize significant system events.

Key words: Computer architecture; functional evaluation; hardware monitor; measurement tools; performance measurement.

1. Introduction

Computer design and performance analysis are two of the three major components of the process which brings computers into being. The process may be represented as a cycle:



where "design" means the specification of function and form in a computing system. Performance evaluation provides feedback which clarifies the suitability of a given design. In turn, performance analyses are constrained to take place within the forms which are available in a given design.

In addition, one of the uses of computers—the third component of the process—is evaluation, the study of computer subsystems for the purpose of local optimization. Proper design enhances this increasingly more significant function. Such local measurements also provide feedback to the design process, since they often show that computers are not as flexible or as powerful as we would like them to be.

2. Contribution of Performance Analysis to Computer Design

Designers select functions according to some model of computing. One very general model was that of Turing's, and it led to a very simple, and abysmally slow machine. Much closer to today's computer was the model which von Neumann had. Although heavily biased by other, contemporary, mathematically-oriented machines, its stored program concept provided a very general facility.

Performance analysis is necessary to quantify models beyond what the designer's intuition can determine. It can also demonstrate new trends in computer usage. Both of these attributes were invoked during the end of the Fifty's when the importance of software development was making itself felt. The fact that over 50 percent of many installations' time was attributable to software development meant that hardware ought to be biased more toward programming. Concepts such as recursion, compatible data representation, virtual memory, time-sharing, and multiprocessing entered into designers' models of computing.

Currently, increasing activity in the file handling and data communications modules of operating systems indicates trends toward data base and distributed computing. If users would document these trends with

quantitative measurements, they could encourage their inclusion in designers' models, which would bring the next series of computers closer to the users' new requirements.

Performance analysis also illuminates the goodness of fit of a particular computer form to common applications. First-generation machines typically had one grossly underdesigned function, floating-point multiply, which was significant to many topical applications. Many program running times were a multiple of the number of multiplications done. (Those days were perhaps the only period when comparative evaluation and predictive performance measures were available; it's ironic that that was due to inadequate computers.) Recognition of the need stimulated progress toward fast multiplications. Unfortunately, this development occurred just as non-numerical uses were becoming paramount.

Analysis of stack machines documented their suitability; such computers allowed simplified compilers to generate code which was as efficient as a machine-language programmer's. Lately, Knuth's analysis showed that compilers spend most of their time doing extremely simple things. However, stack organization should remain in vogue because it anteceded a new trend: structured programming. It is possible to make subroutine calls faster than "go-to's" on a stack computer, which makes the old way in which we have been programming more expensive than the new way without go-to's which we should be using.

A victory of performance analysis toward computer design was the demonstration of superior performance from language-oriented hardware. When the ALGOL-oriented B5500 or the COBOL-oriented B3500 were benchmarked in their respective languages against conventional architectures, they were sometimes able to outperform systems costing over five times as much. This result paved the way for universal-host systems, which can be biased toward one language or another from microsecond to microsecond.

One should understand that the way in which performance analysis effects changes in computer design is not by solid numerical results, but by creating a climate in which specific changes seem desirable or necessary. Proposals for new computers contain performance speculations which are usually disbelieved in their details but which do persuade manufacturers. Documentation of computer design, such as Thornton's book on the CDC6600, typically contain no references to performance analysis.

3. Inadequacies of Performance Analysis for Computer Design

Performance analysis has had its defeats. Designers have often proposed desirable innovations, only to have them rejected due to a lack of corroborating measurements. No measurement yet quantizes the loss of time and resources due to fixed memory size and the poor system design and reprogramming which it causes. Virtual storage was added to computers over ten years ago on an intuitive basis, and still cannot be quantitatively justified, although many users by now have had experience with it.

Most designers' computing models necessarily omit some first-order variables because the scope of the variables extends far beyond the limits of any computing machinery. Until there is a reasonable way to incorporate assumptions about major, non-computing variables which affect performance, predictive measures may be impossible. One such variable is flexibility. For most users, life itself demands that a computer's foremost capabilities be flexibility and ease of change. Attempts at formal definition quickly reveal that these computer capabilities depend on the flexibility of the organization which the computer serves, and that defies definition.

Another characteristic which is of first-order importance is the smoothness with which performance degrades as a computer is gradually overloaded by a growing organization. Even though we cannot do simpler analyses than those of flexibility and graceful degradation, equal time should be given to such salient measures.

Many analysis techniques are difficult to transfer from one system to another. It is important for users to try to make measurements which are either independent of architecture or whose dependence can be measured, in order to build a body of knowledge upon which comparisons can be made between computing systems.

An illustration of the difficulty involved can be seen in the controversy between head-per-track and moveable arm disk. The disk characteristics affect the way a computer system is used, and this is hard to factor out of any measurements.

Users have a capability which designers do not: they can measure the real world. The real world is the only legitimate source of feedback to the designers. Most user groups now provide some statistics to their respective manufacturers, but a systematic program should be prescribed.

It is important for users to accurately characterize the environments in which computing machinery now functions. Of course, characterizations which are system-dependent (such as disk record length) should be stated in a parameterized way (such as, average disk record length=physical disk segment size). High-level language users should be collecting statistics on how they employ such languages, in the manner Knuth's study of FORTRAN did. (The other side of the coin is that every compiler should develop such statistics and merely ask the user to mail them in every quarter.)

Finally, there are some inadequacies of performance analysis which are mainly political. Users should place a high value on built-in measurement tools. Their presence in a system should strongly bias an acquisition decision in the system's favor. Benchmark results should be edited to include probable speed gains due to convenient performance monitoring. Consider two systems, one with built-in performance measurement tools, and the other without. If the instrumented system is twice as slow as the bare system, it may still provide greater throughput and accommodate a larger workload than the bare system because of the inexpensive tuning which can be performed. Normal selection criteria should reflect such a likelihood.

Also, performance measures which are misleading should be discarded from the computer selection process. If a user cannot predict how much work a given machine will do, given its CPU speeds on typical operations, he should not waste his time acquiring the speed information. Performance analysis can help computer design significantly by concentrating solely on the most important criteria, such as flexibility, ease of use, software reliability, and forgetting about traditional measures, such as memory cycle time, CPU operation times, and disk latency time.

4. Contribution of Computer Design to Performance Analysis

In general, the user is in a much better position to take advantage of performance analysis than the computer designer because the user's world is much more constrained. Users can often confine the scope of an inquiry to a decision whether or not to add hardware. Designers spend their time pondering basic questions of system organization, and have small understanding of the impact of one more device to any given configration. Furthermore, the scope of a design revision, covering the entire population of a computer system, leads to cost-benefit trade-offs whose balance point is in the region of catastrophe. A user can make a significant change if he thinks he will realize a 10 percent increase in machine productivity. A designer must be sure all users will realize an impressive gain before he can make a significant change.

The "design-use-evaluation" cycle is completely under the control of the user for his own software systems. For the manufacturer's software and hardware, performance analysis benefits the user primarily to the extent that he can redesign his computer from obtained measurements. Some computer designers have done much to extend the user's capabilities for redesign.

Modularity allows gradual configuration changes in response to slowly changing needs (or in response to equipment failure). The user may have the freedom to add or remove memory modules, processors, peripherals, I/O channels, communication lines, remote facilities, et cetera. If the computer designer has paid adequate attention to flexibility and ease of use, no software changes, either on the part of the user or the operating system, are required as part of a configuration change; new units can be fully utilized as soon as they are brought on-line.

User-modifiable microcode allows gradual changes in processing or I/O function. Effective use of this feature depends on a detailed understanding of what the user needs most. In many cases, the paramount need is to relax artifical limitations which machine architectures impose, such as maximum main memory size.

Cross-bar exchanges allow gradual changes in the connectivity of a computing system. Their administration should be governed by measurements of traffic along existing connections.

Since many systems are limited by operating system response times, it is often necessary to specialize an operating system to one's installation. Operating-system-generating programs are one tool which designers can plan for and make available for creating special-purpose operating systems.

User control over high-level language program structure permits programs to cooperate gracefully with whatever storage management scheme has been implemented. It is to be expected that software designers who provide such flexibility also provide the measurement tools with which to obtain the information which is needed to use it well.

System accountability, through log information of specifiable detail, is another designable property that can give the users information they need to extract the maximum throughput from their computer. Reports of CPU, device, and software utilization frequently cannot be generated from systems whose designers ignored accountability.

5. Inadequacies of Computer Design for Performance Analysis

In general, designers should not prohibit measurements. When computers were simple, many measurements were trivial because important events in a system's behavior caused unique hardware phenomena. For example, system overhead could sometimes be obtained from the amount of time the computer's program counter was less than a particular value. Now that systems are forbiddingly complex, it is the responsibility of designers to provide the unique hardware phenomena which make measurement simple.

One technique which adds about .001 times more cost to a system is the "monitor no-op" or functionless machine instruction which has many bit representations. For example, if the ordinary no-op has unused bits, and if the unused bits may contain any pattern of 1's and 0's, one can distinguish when a system is executing various parts of its operating system if these no-ops, with unique patterns, are scattered in judicious places. Naturally, software systems, as well as user programs, should be able to generate unused monitor no-op patterns, too. This makes an integrated software-hardware measurement system trivial.

Whatever measurement tools appear in computers are typically just those which the designers themselves use. Without constant exposure to user requirements, additional tools are usually eliminated during cost-cutting programs. If the designers are only responsible to the manufacturer for hardware, measurement tools are minimal or nonexistent. On the other hand, if the designers are responsible for hardware, firmware, software, and intersystem connections, all manner of tools will be built in.

At the very least, important hardware monitor probe points should be accessible from the console. It is also a prerequisite for allowing plug-in monitors, which can be taken from one computer to another of the same model population and hence establish uniform measurements for that model. It is also a step toward various levels of sophisticated monitors, which are capable of progressively more detailed reporting of measurements, and which allow each user to become involved in performance analysis to various degrees.

Taking this one step farther, designers should be encouraged (by statements of how much users are willing to pay) to make monitoring modules available as options in a system. One can go from a bank of ammeters to histogram displays, to weekly reports on subsystem utilization and heavy users, and to dynamic scheduling suggestions to the operator.

6. Summary

Performance analysis provides substance for the conceptual models which designers use to generate new systems. Most of the major variables, however, are still unanalyzed. Computer design can provide or prohibit both the information needed by performance analysis and the redesign of systems or configurations for greater productivity. Hardware which assists with information gathering can be added very cheaply, but the need for it has not gotten through to most hardware designers and manufacturers.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION;
REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Validation Aspects of Performance Evaluation

Raymond J. Rubey

Logicon, Inc., Dayton, Ohio 45432

This paper describes the relationship between software validation and computer performance evaluation. A brief review of validation objectives and methods is presented. With this background, three principal aspects of the relationship between validation and evaluation are explored.

The first aspect to be explored is the activity undertaken during validation to compare the actual system performance with performance predicted earlier. The second aspect, with which the paper is concerned, is the difficulty of validation. This should be an important consideration in the evaluation of a particular software or hardware system. The third aspect of the relationship is the similarity of the tools used in validation and performance evaluation.

Key words: Actual system performance; computer performance evaluation; performance prediction; simulation; software validation; tools.

1. Introduction

Software validation has become as important an activity in many computer-based system development activities as the programming activity. The relationship between software validation and computer system performance evaluation is therefore an important subject for study. The overall objective of a software validation activity is to convincingly provide assurance that a computer system will, in combination with the given hardware, satisfy all user requirements. In a narrow sense this often is interpreted as requiring a demonstration that there are no errors in the program that will keep it from producing correct outputs for all reasonable input data. However, a complete validation activity has a much greater scope than a continuation of programmer debugging. The relationship between validation and performance evaluation is apparent when the complete validation activity is considered. The next section has a brief discussion of an idealized validation activity. This discussion is based on the validation of real-time aerospace software; both because of the experience of the author and because it is in the development of aerospace software systems that validation has been most widely and successfully employed.

The validation of real-time aerospace software has received considerable attention because of the potentially catastrophic consequences of an error in aerospace applications. Many aerospace software customers, recognizing that failure to validate operational software is false economy, are willing to devote the attention and resources needed to achieve the requisite highly reliable software. For these reasons aerospace software validation has reached the level where other application areas can profitably borrow from its concepts and methods.

2. Validation Methods

The idealized software development cycle proceeds in distinct stages. First, system requirements must be determined and described in detail. Next, algorithms which satisfy these requirements are developed and described in the software specification. A detailed program design is then created from this specification. Finally, the program is coded in accordance with this design.

As noted earlier, one important facet of validation is the comparison of the program's capabilities with the system's requirements. However, experience has shown that it is not desirable to wait until the pro-

gram is coded and debugged to begin the validation. Waiting until the final stage to begin validation can result in a program which is so unintelligible, poorly organized, or intricate that validation is impossible within reasonable cost and schedule constraints. Software problems may be discovered that have their origin in the software specification or detailed program design. Such problems should have been discovered earlier and, if they had been, an appreciable savings in development effort and time would have resulted. Therefore, a successful validation activity begins with an examination of the system requirements, and then, in sequence with the software development, proceeds to a determination of the software specification's ability to meet these requirements; then to a detailed comparison of the program design with the software specification; and finally, to a verification that the computer code implements in every aspect the detailed design. By verifying the correctness of each intermediate product of the software development process, with the preceding product as the reference, the ultimate task of matching the final program with the system requirements is greatly facilitated.

The cost of such an incremental verification of system correctness is more than balanced by the savings that accrue because errors are detected early. The validator makes extensive use of simulations in performing each incremental verification step. The early simulations used to verify the algorithm's ability to satisfy the system requirements are often crude and imprecise, however, as the validation process continues the simulations are refined to make them more accurate representations of the real environment. In some cases this elaboration of the simulations continues to the stage that virtually all of the systems components are represented by the actual hardware.

One software performance characteristic that must be verified at each stage in the software development process is the ability of the final program to perform the required computational tasks, using the existing hardware configuration, in the time available. This task is relatively straightforward after the final program has been developed. However, in the earlier phases of validation one must rely on an imperfect knowledge of hardware and software performance. It is at this early stage that the relationship between validation and performance evaluation first appears, in that the validator must verify that the expected performance of the hardware system meets the expected computational, input/output, and storage needs of the software. The greater the confidence that can be placed in these estimates, the smaller the margin in excess computational, input/output and storage capacity the validator can accept.

3. Validation Performance Estimation

How are these estimates of hardware and software performance obtained by a validator? A validator often uses methods that are no different from those that the system and software designers have used to select their design approach. In some cases the validator double-checks the designer's performance evaluation; in other cases he extends the designer's performance models to represent critical system components in more detail without changing the basic structure. If the validator judges that the designer's performance evaluation was inadequate, a completely new evaluation may be performed.

The validator's performance evaluation task is more circumscribed than that of one who is concerned with comparing two computer systems. The validator evaluates the capability of a specific hardware system to perform specific software functions. There are additional complications in real-time aerospace software as compared with scientific or commercial software. Aerospace software must satisfy stringent response time requirements; times required between a stimuli and a response on the order of 10 milli-seconds are typical. These response times must be achieved with a limited computer capacity; frequently the software requires over 90 percent of the available computer memory and time for execution.

These constraints require that the performance evaluation produce answers with high confidence albeit over a restricted range. Simulation is the most powerful validation-oriented performance evaluation technique. It is employed if the expense of a sufficiently precise simulation can be tolerated. To supplement simulation, timing and sizing estimates for each software module are prepared in terms of the basic computer operations (e.g. load, add, multiply, disk access). Time line analyses are performed in which the period for the execution of each software module and input/output operations are plotted on the scale of available time. Determination of the worst-case execution times for alternative program paths is accomplished by a PERT-like network analysis. As such performance

evaluations are performed the validator must compare the results against the requirements, specification or detailed design; taking into consideration the uncertainties in the evaluation results. To gain some insight into how accurate computer memory requirements can be predicted, the original estimates and the actual memory utilization for a large real-time avionics software system were compared. The original estimate and the actual memory utilization were identical; both were equal to the size of the memory that was available. This obviously tells one little about the accuracy of the estimation methods although it may tell one something about the ability of programmers either to squeeze a program into the available space or fill up the space available. For the application examined, it was the former situation because when estimated and actual sizes of individual modules of the program were compared the differences were as great as 50 percent. What actually happened in this project, and in many others, is that modules were rewritten many times to get them to fit, with some modules being smaller than the original estimates in order to balance those modules which grew and could not be reduced.

The use of a higher-level programming language adds another factor to the timing and sizing analysis and subsequent performance evaluation. It is generally recognized that the use of a higher-level language and its associated compiler result in less efficient machine code; unfortunately, there is little quantitative data on the extent of this inefficiency. One approach that has been taken in specific projects is to calibrate the compiler. This is accomplished by coding benchmark programs in assembly and higher-order languages, assembling and compiling both versions, and comparing the resultant object code. One pitfall of this method is the possible differences between the benchmark program and the application programs with regard to the types and frequencies of the functions to be performed. Another is the considerable differences in programmer performance that, for example, have resulted in one programmer's version requiring twice the memory as another's even though both used the same higher-order language and worked from the same program design.

The lack of attention paid in scientific and commercial programming environments to higher-order language and compiler inefficiencies is due in large part to the widely accepted opinion that any such inefficiencies are more than compensated for by greater programmer productivity.

The small amount of knowledge that exists concerning a particular compiler's inefficiencies is symptomatic of the relative lack of truly effective methods for determining the computational requirements of software before that software is developed. Considerable effort may partially compensate for this lack in the aerospace environment but even then there always exists the possibility that substantial system redesign late in the software development process may be required because of inaccurate performance estimation.

4. Validation-Oriented Constraints

In developing a computer system, the effect of the hardware and software design alternatives on validation must be considered. It is possible to define a hardware/software configuration which cannot be adequately analyzed or tested during a reasonable validation process. Even without reaching this extreme, particular hardware and software alternatives which at first appear to be the cheapest may become the most expensive when the impact on validation cost is considered. For example, it has been claimed that validation costs for the Apollo guidance software were doubled because of the absence of hardware floating point instructions and the absence of a single instruction "save for restart" capability in the Apollo Guidance Computer and because of the complexities of the Apollo software's interrupt executive.

Validation imposed constraints may actually reduce the effective computational capacity of a computer system. This often occurs in real-time aerospace systems even though, as was mentioned earlier, high efficiency is more important for such systems than for most scientific or commercial applications. Correct software behavior can and usually should be given precedence over high efficiency.

One example of the trade-off between efficiency and ease-of-validation that has been made in several aerospace systems involves the design of the software executive or operating system. As in scientific and commercial systems, the software elements or modules which do the computational work have their execution scheduled, invoked, suspended and terminated by the executive.

Different modules have to be executed at different rates, some periodically and some upon demand. An interrupt-based executive, using the interrupt facility of the real-time computer, suspends the execution of a low priority module when the indication is received

that a higher priority module should be executed. Then, after the high priority module has been executed, the execution of the low priority module is resumed at the point where the interruption occurred. The validator must demonstrate that the interruption of any module at any point in order to execute any higher priority module does not affect the correctness of the results obtained. This can be very difficult when there are many interrupt levels and when the modules share a common data base. To alleviate this problem, some aerospace software executives have been designed so that each module is short enough to permit it to be executed before any interruption can occur. Each module returns to a do-nothing routine after execution; only this very simple routine is interrupted to execute another task. This type of executive requires that the software be divided into small "bite-size" modules. It also wastes computational capability because the do-nothing routine is often being executed instead of some useful module. The performance of a system having this type of executive will often be considerably less than the performance of an identical system that has a conventional interrupt executive.

5. Validation-Oriented Simulations in Performance Evaluation

The use of simulations in validation has already been mentioned. At the final stages these simulations can be very elaborate and detailed. For example, simulations of a real-time computer system on a large scale general purpose computer are often employed. Such computer simulations are capable of duplicating, bit for bit, the actual behavior of a real-time computer in executing the aerospace software. The computer simulation operates in conjunction with an equally detailed simulation of the environment which produces the inputs for the real-time computer and acts on the outputs that are generated.

If information about the performance of an existing system or a modification of that system is desired, the simulations that were designed for validation provide a powerful tool for obtaining that information. Such simulations are usually much more heavily instrumented than are hardware monitors used in performance evaluation. Such simulations already have probes that allow access to system parameters, such as the real-time clock, and provide mechanisms for accumulating the data from these probes and outputting this data in user defined formats.

By use of a computer simulation it is simple to measure the time required for execution of any software segment, the total time that is spent performing any particular computer operation, and the time between any stimuli and the corresponding response. The time measured is the time that would be measured if one had the same access to the actual system and not the simulation's execution time; typically such simulations run much slower than real-time.

Of course much of the information that can be obtained has already been gathered during the validation in demonstrating that the program satisfies the system requirements. By making appropriate changes in the simulations it becomes possible to measure the effect of modifications to the actual system. For example, by changing the appropriate parameter and then exercising the system through simulation one can evaluate the effect of a faster memory on system response time.

In the past a validation-oriented simulation was designed and implemented for one particular hardware system configuration with little thought to creating a general capability useful for the simulation of many different configurations. Recently much more attention has been paid to developing more general simulations, largely because of the desire to reduce the cost of future systems. For example, computer simulations have been developed in which the user can specify the word size of the real-time computer. To determine the optimum word size for a particular application, the user runs several simulations, each with a different word size, and selects the word size that provides the needed accuracy.

6. Summary

Validation and performance evaluation are complementary pursuits. Improvements in performance evaluation methods will be of benefit to validation activities. The existing validation methods for predicting computer system performance, although far from perfect, should be of interest to anyone interested in the applications of performance evaluation. A performance evaluation should not neglect validation considerations, particularly where inefficiencies must be tolerated in order to facilitate validation. The simulation tools used in validation can provide data for performance evaluation, especially when modifications to an existing system need to be evaluated.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION; REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Security vs Performance *

Dennis R. Chastain

U.S. General Accounting Office, Washington, D.C. 20548

The necessity for security often overrides the concern for optimum performance of a computer system. However, it is important that the relationships between security and performance be recognized. In this paper three major areas concerning these relationships are discussed.

The first concern is with some of the types of hardware and software that are required in order to maintain security internally in an ADP system, and the effect of this hardware and software on the performance of the system.

The second area discusses some of the complex problems of evaluating the impact of security software on the performance of computer systems.

The final area discusses a number of other technical and human problems often associated with evaluating performance in a secure environment.

Key words: Computer performance; computer security; hardware monitors; performance evaluation of secure computer systems; security, data transmission; security, file access; security, identification (password); security, input and output processing; security software; software monitors.

1. Security Hardware and Software

1.1. Special Security Hardware

Security hardware is generally employed when computer systems must transmit and receive sensitive data over telecommunication lines. This type of hardware is employed to protect data during transmission. The hardware consists of cryptographic devices at each end of the communication line. Basically, a device at the originating end of the communication line scrambles data into some seemingly meaningless bit pattern. The device at the receiving end of the line unscrambles the bits into the original form. This security hardware usually operates at speeds at least as fast as the transmission speed of the communication line, and therefore causes no degradation in the performance of the computer system.

—File Access Security,
—Input and Output Processing Security, and
—Data Transmission Security.

These three are discussed below with some comments on other internal security precautions which may be considered.

1.1.1. File Access Security

This level of security basically is a check at "file open time" to determine if the user attempting to open the file for processing has permission and clearance to access the file. Usually the list of users, their access rights, and their clearance level is stored in a direct access storage file. When opening a file, the user's record is pulled from the list and examined to determine if he has permission to access the desired file. A comparison is also made of his security clearance and the highest level of security data in the file.

This level of security imposes insignificant overhead on the system. The time spent opening files is usually minor relative to the time spent processing the data in the files.

The processor time required to make the security level and permission determination is insignificant.

The number of additional input or output instructions required to make these determinations varies depending on the number of users with access to the system, the number of protected files on the system, and the blocking efficiency of the files which contain the access lists. In one such system the additional I/O's required are usually less than ten.

1.1.2. Input and Output Processing Security

This is the lowest level of software security. It involves security checks on every input or output instruction received by the system. There can be several security checks made at this level. The most elementary check is to insure that the user attempting to access the data is the one who was processed through file access security at file open time. A check is also made to insure that the user has the authority to do what he wishes with the file. For example, a check may be made to insure that a user who has "read only" authority is not attempting to alter information in the file. Additional security checks could be made at this level.

In the system with which the author is familiar, the overhead introduced at this level can be as low as one-tenth of one millisecond per input or output instruction.

1.2. Special Security Software

Unlike security hardware, security software does cause some degradation in the performance of computer systems. The degree of degradation depends on the level of security which is employed (as well as the efficiency of the code). There are at least three levels of security which can be implemented internally in a computer system:

1.2.1. Software Data Transmission Security

The security hardware discussed previously only "protects" data while it is being transmitted. It does not determine whether or not the data should be transmitted. This determination is normally made by an additional software security check. The purpose of this check is twofold. First, it insures that the security level of data to be sent is not higher than that authorized to go to the destination terminal. Secondly, it insures that the security level of the data is not higher than that authorized to be received by the person who is using that terminal. Security of this nature is usually implemented in the telecommunication software, whereas the levels previously discussed are usually implemented in the input and output control system of the supervisory software.

In a system familiar to the author the overhead introduced by this level of security is less than one millisecond per transmission.

Within the author's experience in measuring performance of secure systems, a properly designed system should not add overhead greater than 5 to 10 percent of any resource for these three levels of software security. On one such system the overhead is less than one percent.

Additional overhead may be introduced if there is a requirement, at one or all of these levels of software security to produce sufficient information to accommodate a security audit trail.

1.2.2. Additional Internal Security Precautions

There are at least two other types of overhead that are associated with most secure systems.

First, there is overhead caused by requiring security identification information to be printed at the top and bottom of each output page on either printers or terminals.

Secondly, there may be some special hardware or software associated with computer systems which is designed to occasionally test or attempt to subvert the security of the system. This function tests the integrity of the security system by occasionally attempting to perform the operations that the security system is designed to prevent. In one system it was found that this function, due to the frequency of its use, was taking seven percent of the total processor time. (This problem has since been corrected.)

Depending on the degree of security required by an installation, additional supervisory software may be implemented to provide more comprehensive internal security in computer systems. Some functions this additional software could perform are:

—special validation of programs requesting to enter supervisory mode,

—monitoring and validation of certain types of requests within supervisory mode—such as requests to access different locations in internal memory,

—prevent access to the control information of files (as opposed to access to the data in the files),
—periodically test the system to insure that the mechanisms established to control applications programs have not been illegitimately altered, and
—to destroy sensitive data when it is no longer needed by writing over the files with meaningless data.

The additional overhead caused by any of these functions depends on how it is designed and implemented in the operating system and the efficiency of the programming required to perform the function. The function of writing over files to destroy the data could introduce significant I/O overhead since many applications and utility programs utilize a significant number of temporary files.

2. Performance Evaluation of Secure Systems

Security routines are usually part of the supervisory software. The security software provided by the computer vendors may not be sufficient to satisfy the unique security requirements of individual installations. When this situation arises, modifications—which may not be minor—must be made to the supervisory software.

The task of measuring the effect of security software on the overall system can be a difficult assignment. The problem is essentially the same as that of monitoring any non-standard supervisory software. Three techniques are especially applicable to measuring the effect of security software on the overall performance of the computer system:

—advanced hardware monitors
—detailed simulation models
—software instrumentation.

A hardware monitor which can monitor activity at selected memory locations can be used to ascertain overhead due to security. Several existing monitors have this capability; however, they are somewhat expensive.

Detailed simulation models may also be used to determine the effect of security software. Since security software usually consists of modifications to standard supervisory software, traditional computer simulation packages (such as SCERT and CASE) can not be utilized. Models of the detail required have to be constructed in lower level simulation languages such as SIMSCRIPT, SAM and ECSS.

The construction of detailed simulation models in lower level simulation languages requires considerable expertise, time, and computer resources. Modeling of supervisory software is difficult. It requires the same level of understanding as that needed for the original writing of the software. Considerable computer resources are also required to debug, test and run the models. It would not be inconceivable for a detailed model to have an actual run time to simulated time ratio of 10:1.

Software instrumentation is basically a monitor which is in continuous operation, for it is built in as an integral part of the supervisory software. Software instrumentation consists of specially located instructions that count and time the execution of the security software. The count and timing data is used to evaluate the effect that the security software has on the overall computer system.

Since software instrumentation is imbedded in the operating system, programming for it, like most systems programming, is a very difficult task. This programming task must be repetitiously performed for each new release of the operating system. Because this coding cannot easily be removed it continually imposes a degree of overhead on the computer system.

3. Technical and Human Considerations

When making performance measurements in a secure environment, additional technical and human factors should be considered.

The popular technical factor to be considered is, "Can a performance monitor obtain sensitive information from a secure computer system?" The discussion of this issue involves two considerations:

—accidental compromise of sensitive information, and
—intentional compromise of sensitive information.

3.1. Accidental Compromise

It is unreasonable to suspect that hardware monitors composed of only mechanical and electronic counters could inadvertently obtain sensitive information from computer equipment. Hardware monitors which can monitor data (i.e., bit patterns) in addition to just

counter data could possibly record sensitive data. However, it is doubtful that monitors could accidently be attached to the computer in such a manner that meaningful alphanumeric data would be recorded. In addition, the data reduction software that is used to process the monitor data into reports would probably be unable to process data that has been erroneously recorded.

It is conceivable that software monitors could record sensitive data. Primarily, this is because most software monitors operate in supervisory mode and have authority to access data anywhere in the computer system. However, the author does not believe it is likely that meaningful data would be recorded inadvertently, and even though data were recorded it would not be likely that the data reduction software supplied with the monitor could process the erroneous data.

3.2. Intentional Compromise

Using monitors to intentionally obtain sensitive information from computer systems is quite another matter.

It has often been noted that the applications of hardware monitors are limited only by the creativity of the individuals using them. Today there are in existence some very advanced hardware monitors. Some of these not only have internal storage but are actually minicomputer based. Such hardware monitors could record some of the alphanumeric data that is being processed by computer systems. Data reduction software could be developed to process the recorded alphanumeric data.

Another security consideration concerning hardware monitors (or any hardware device) is the possibility of a foreign device being placed in the monitor to transmit the conversations of the people in the computer area or electronic signals from the computer. Some monitors are physically complex internally and it would be difficult for the layman to detect such a covert device.

Since software monitors operate in supervisory mode and have the complete system at their disposal (unless prevented by special security software), it is entirely possibile that they could obtain sensitive information from the system. Again it would not be difficult to implement reduction software to process the recorded data.

3.3. Human Problem

The number of people who have access to a secure computer facility is usually limited. The backgrounds of these individuals usually undergo a comprehensive investigation to determine if they are likely to compromise sensitive information to competitors (or in the case of the Government, to subversive organizations).

Many performance measurement tools and techniques in use today are acquired from outside vendors. If problems arise (don't forget Murphy's Law) vendor personnel may have to be called in to find and solve the problems. If their backgrounds have not been investigated, special controls must be followed before they are allowed entry into the computer facility. As a minimum, they will have to be escorted by authorized personnel. Depending on the sensitivity of the data, all normal processing may have to be terminated until the uncleared personnel have corrected the problems and left the premises. This may sound like a minor inconvenience, but installations which are new to performance evaluation may depend heavily on outside support.

4. Summary

The necessity for security is often a greater concern than the performance of the computer system. However, it is important that the relationships between security and performance be identified. The specific relationships identified in this paper were:

—Cryptographic hardware has no effect on the performance of the system,

—Security software should not degrade performance by more than 5 to 10 percent,

—Determining the degree to which security software degrades performance may be a complex and costly task,

-It is not likely that performance monitors would unintentionally compromise sensitive information from secure computer systems; however, it is possible for performance monitors to intentionally obtain sensitive information, and

—If utilizing performance monitors one should consider the inconvenience of occasionally tolerating uncleared personnel in the immediate vicinity of the secure computer.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION;
REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Performance Evaluation Techniques and System Reliability—A Practical Approach

James Hughes

Xerox Corporation, El Segundo, California 90245

Abstract

A literature search discloses very few papers devoted to the improvement of system reliability through the use of performance evaluation techniques. A brief description is provided of an existing hardware monitor of advanced design which is capable of discerning both software and hardware events. In terms of such a tool, methods are discussed by which an attack may be launched on a number of the root causes of system unreliability. In order that new forms of packaging technology may not jeopardize the continuing use of such techniques, a proposal is made for the inclusion of a monitoring register in future computer systems.

Key words: Hardware monitoring; monitoring register; software monitoring; system reliability.

1. Foreword

"While the JCC's attempt to present the specific results of recent research, the Workshop's objective is to integrate research results with practitioner's experience to determine the areas in which performance evaluation techniques and facts are currently well established, and the areas in which further research is required to develop adequate techniques. The Workshop papers should identify significant facts, attempt to develop taxonomies, and initiate fruitful discussion, while the JCC papers are usually stand-alone descriptions of work."

Taken from a pre-Workshop guide to participants, the quotation which stands at the head of this paper gives the ground rules which were observed by authors in planning their contributed position papers.

Unhappily, the use of the software or hardware-monitoring techniques of performance evaluation is far from well established insofar as practical improvements in computer system reliability are concerned.

The author will proceed to justify that statement in a moment, but first invites the reader to consider what such a situation implies. According to the ground-rules, he must nevertheless "* * * integrate research results with practitioner's experience * * * must determine areas in which further work is required * * * should identify significant facts, attempt to develop taxonomies, and initiate fruitful discussions * * *"

This author was determined to discuss actual achievements rather than proposals, and to concentrate upon the practical rather than the theoretical. He was, therefore (in the absence of any body of work published in this field), forced to fall back on his own "practitioner's experience," which was not directly aimed at reliability improvements. Rather, that experience was concerned with a development project aimed at the improvement of current techniques of fine-detailed hardware and software system measurement, and to be carried out with minimal artifact on a working computer system. It so happens that the capabilities of such a tool are precisely what are required in order to be able to apply performance monitoring techniques to the improvement of system reliability. This position paper, therefore, of necessity will conform more closely to the description of a JCC paper than we might wish; the author trusts that it will at any rate "identify significant facts, and initiate fruitful discussion."

2. The "Non-Establishment"

Returning to the paucity of published work in the field of performance evaluation techniques devoted to improvements of system reliability, a literature search was made of the main stream of published computer literature. It was considered that the JCC papers of 1971 and 1972, together with the references to other work contained therein, were representative of work undertaken in the past five years, together with previous publications which were in some measure landmarks and had continuing interest.

The criteria by which published works were selected were as follows: list A was formed containing papers devoted to performance evaluation involving software or hardware monitoring techniques; list B contains papers covering system reliability; list AB represents the intersection of lists A and B, namely, papers which involve both performance measurement and system reliability. Now list A contains 52 papers of which 20 are significantly hardware-oriented. List B contains 71 papers, of which 43 are significantly hardware-oriented. List AB contains five titles, just one of which concerns hardware methods for monitoring (real-time) systems in order to improve reliability.

It may be of interest to note that a further 85 papers which involved performance evaluation and/or simulation were considered, but were excluded on the grounds that they did not in any way touch upon real measurements of the systems being discussed, but rather concerned modeling or mathematical analysis and discussed performance evaluation only in a general way. The five papers contained in list AB are given in Appendix 1, and inspection of the papers shows that several only narrowly made the list.

Paper AB1 is typical of several papers describing evaluation of user programs, generally written in FORTRAN or COBOL. It was included here because one class of errors which it detects is attributable to quirks in implementation of the object system arithmetic unit. Papers AB2 and AB4 concern error recovery through software, and come close enough to usage of software monitoring techniques to justify their inclusion.

Paper AB3 is of considerable significance, since it parallels much of the work to be described in the present paper, but in the area of critical real-time systems.

Paper AB5 was included on the grounds that it covers, although in general fashion, precautions taken with a user-oriented transaction system to ensure adequate performance and high reliability.

3. Possible Explanations

It is clear from the size of lists A and B that the two fields have enjoyed very considerable interest among computer professionals during the last few years. Why, then, this surprising state of affairs, where so little evidence is shown of attempts to adapt techniques from the one field to the requirements of the other?

Perhaps we should examine the problems which face the would-be practitioner. Accepting for the moment the continuing dichotomy between software- and hardware-oriented performance measures, let us first consider the difficulties which beset an exclusively software monitoring approach.

First there is the problem of reliance upon the suspect system, which is by definition of dubious reliability. Second, there is an inherent difficulty in all forms of serial software monitoring, namely that of making a set of adequate observations without causing an intolerable degradation in system performance. Third, there is a problem of accessibility at a level of fine detail to those items of information (depending on the particular system architecture) which are inextricably embedded in the hardware.

Despite these factors, there do exist some examples of software monitoring for improved reliability, particularly in the area of retriable I/O functions.

Turning to hardware monitoring techniques, superficially it would appear that this approach offers a greater potential for reliability improvement because it possesses several inherent advantages.

First, there is independence from the subject system and its suspect hardware and software. Second, there is an implied ability to tap any of the signals which exist in the wiring of the subject system's backpanel. Third, there is a wide range of resolution in time and space of the signals which are of interest, in contrast with the overhead and accessibility problems of the alternative approach.

On the other hand, there is a serious drawback residing in the inability to observe and discriminate upon software events, and thereby modify, in some way, the purely hardware measurements.

The question must still be faced: why has no serious attack on system reliability through this avenue been reported?

A well-considered answer to that question appears to involve several factors. There are the technical problems of knowing where to look, of validating the accuracy of a nontrivial number of temporary attachments to backwiring, or certifying (on an unreliable system, remember!) that the attachments themselves do not compound existing problems, of planning and implementing a manual setup of the hardware monitor's controls (including patch-cording a special plugboard, in all probability), and finally, of interpreting the captured data. Small wonder that the software specialist prefers to pore over voluminous 'crash dumps', while his hardware-oriented colleague peers into his oscilloscope.

Is there a way of resolving this dilemma? The author believes that there is; it necessarily involves some rethinking of the conventional acceptance of a cast-in-concrete division between software and hardware monitoring. In fact, we must consider ways and means of merging the two techniques so as to maximize their conjoint effectiveness.

This "practitioner's experience" with such a system will shortly be published in a paper entitled, "On Using a Hardware Monitor as an Intelligent Peripheral."[1] In that paper, the co-authors describe what they believe to be the basis of the hardware/software monitor of the future. In summary, it involves a high performance programmable hardware monitor, a means for selecting and conditioning a desired subgroup from a large set of permanently available hardware signals, and a means of initiating the transmission by the subject operating system of encoded identifiers specifying software events. Given this configuration, the entire monitoring subsystem may be typified as an intelligent peripheral device forming an extension to a host system, rather than a mere inflexible appendage to the subject system under observation.

4. Areas of Potential Application

In describing the application of the enhanced hardware monitor to the improvement of system reliability, we shall attempt to group areas generically in an attempt to fulfill our commitment to 'develop taxonomies.'

[1] J. Hughes and D. Cronshaw. "On Using a Hardware Monitor as an Intelligent Peripheral." Performance Evaluation Review, ACM (SIGME) Vol. 2, No. 4, December 1973.

5. Antecedence (also Posteriority)

The well-known trace is what we have in mind. It is possible, however, to accommodate a number of interesting variations:

1. Traces leading up to, as well as following, an event.
2. The event may be of a hardware, software, or hybrid hardware/software type. For example, the execution of a specific instruction by a specific user, while in a specific hardware state.
3. The trace may be of all instructions executed, or of a selected subset of instructions (conditional branches, perhaps), or of data reads or writes at specific memory location(s); it may include software data (e.g., identity of current CPU user).

A trace incorporating some of the above elements is shown in Figure 1; a full explanation of its significance will be given shortly.

6. Localization

Here we would be concerned with a situation where it is suspected that corruption from an unknown source is affecting critical locations in memory, which contain either data or code. (In the latter case, a crash dump would, in all probability, disclose the exact symptom, and the next application area "Provenance" would be called for.)

A case in point might be the inadvertent overwriting of cells in a system resource allocation table, indicating free and allocated blocks of pooled file storage space. An equally serious situation might involve catastrophic changes in critical code stored in trap or interrupt locations. Figure 2 shows output for our programmable hardware monitor showing the sequence and the frequency of overwrites into a critical code area.

7. Provenance

Here we would be concerned with a question such as "which system task provoked this error?" Assuming that the error is discerned by our monitor, then tagging it with the appropriate identifier (provided by the subject operating system) is comparatively simple. Returning to Figure 1, we have an example of this type of situation. The error to be detected is of a complex nature. It arises because any of a number of sys-

tem tasks is permitted to embark on a time-slice with interrupts disabled. By convention, however, all such tasks must re-enable within an allotted time, let us say less than 300 microseconds. Suppose that we have evidence suggesting that one or more tasks is contravening this rule? Then our hardware monitor may be utilized to detect the situation. The output data shows for how long the disabled state had persisted when intercepted and also the user identifier number (line 01). It also continues with a trace of consecutive instruction locations and opcodes, together with the states of the interrupt inhibited and privileged instruction flags (lines 02 to 30).

8. Prolongation

This application arises in real-time systems where we wish to improve reliability by ensuring that the range of periods of time when interrupts are continuously inhibited remains below some desired upper limit. The primary task is to identify and order the long disable sequences. It then becomes possible to form a judgment as to whether improvement (by separating non-reentrant sequences with a disable/re-enable breakpoint) is feasible, and if so, precisely which sequences must be reduced.

An example is provided in Figure 3 which shows a ranked list of identified disable sequences in the Sigma UTS Operating System (release C01). In this case, since some of the disabling points arise in overlays of the operating system code, supplementary runs (of the "provenance" type) are required to positively identify those members of the set.

9. Distribution

The associative memory of our proposed hardware monitor has some interesting properties when used as an interval timer; that is to say, when precoded with data representing time interval values and then requested to match an input (time) value.

If there exists a possibility that outlying interval times may exceed the maximum preloaded value, it is possible to avoid the equivalent of the "frequency folding" problem which plagues aerospace instrumentation by modifying the data reduction algorithm as follows. One cell of associative memory (e.g., line 61 in Figure 4) is uniquely preloaded so as to guarantee a mismatch on initial inquiry. An outlying value failing to match at the first attempt is then forcibly matched with this cell, a frequency count is incremented, and the current (outlying) value is compared with the current maximum outlying value, and the greater of the two written back into data storage (MAXIMUM, line 61). In this way sufficient information may be gathered to form a necessary and sufficient description of the statistical distribution of the periods which are of interest.

10. Utilization

A latent source of unreliability in a supposedly fully tested and debugged operating system may reside in the fact that some sequences of code have never been exercised and, therefore, are not validated. In the presence of some extreme condition, at a future time such code may be activated for the first time—under critical working load, with unpredictable results.

Our proposed hardware monitor may be utilized to subject all system code to close scrutiny—and unlike previous hardware monitors, may cover overlays and system-initiated "user" tasks, as well as permanently resident code.

Figure 5 shows an example where lines 6 through 8, representing instruction address X'3D00' through X'3D2F' are never executed in either of two possible modes.

11. Selective Fault Injection

Another possibility—which comes close to what J. Gould describes in Reference AB3—is to stimulate the subject system with an injected error or overload condition, then monitor the resulting reactions, using techniques described under previous application headings.

Some such technique is possible with existing hardware monitors, operated manually. A much more powerful insight is possible with our proposed system, however, not merely because of improved facilities for observing resulting software/hardware events. The payoff would come from the programmability of the monitor, making it possible to pre-select vulnerable intervals during which one would like to have the error condition injected.

12. Problems and a Solution

This may have appeared to be an overly optimistic projection of what is possible with existing equipment. In contrast, what problems may be foreseen?

Among hardware monitoring practitioners it is well known that each make and model of computer presents idiosyncratic difficulties. These typically are concerned with such matters as instruction pre-fetch before or during conditional branches, observation difficulties for opcodes executed from registers or from interrupt locations, skewing in time of signals, such as opcodes and success/failure when the code was a conditional branch, etc.

These difficulties create relatively minor problems leading to negligibly small inherent errors with current equipment. New equipment now coming to the field is liable to compound some of these types of problems and also to add a new one. With new types of packaging, access to signals of interest may well become prohibitively difficult as compared with attachment to most computers in use today. Unless a solution can be found to these problems, then sophisticated forms of joint hardware/software monitoring—with all the advantages which they offer to performance evaluation—may never come to maturity.

The only solution appears to be for computer manufacturers to include what may be called a Monitoring Register in their equipment, and provide standard plug and socket accessibility for external equipment.

Appendix 2 sketches a format for such a register, arranged with four optional levels of implementation.

Option I would provide all that is needed for simple low-cost (possibly graphical) resource overlay monitors.

Option II would go beyond that and enable execution intensity by address, opcode mixes, or simple event observations to be made.

Option III would extend to the current level of performance described in this paper, by including software event identification.

Option IV would make possible, in an elegant manner, a two-way interaction process between system and Monitor. This leads directly to possibilities such as the optimization of scheduling, early warning for overload reconfiguration, and the like.

13. Summary

This paper has outlined what is possible for reliability improvement using existing techniques of joint hardware/software monitoring. (Indirectly, perhaps, it has also indicated new fields for general performance evaluation.) The opportunity has been taken of raising a warning voice on the subject of the increasing difficulty of performing hardware monitoring, given the increasingly complex and inaccessible packaging technology of the new computer systems, and a proposal has been made which would mitigate many of these difficulties.

APPENDIX 1

[AB1] H. S. Bright, B. A. Calhoun and F. B. Mallory, "A software system for tracing numerical significance during computer program execution," Proc SJCC 1971, AFIPS, pp. 387–392.

[AB2] D. D. Droulette, "Recovery through programming system/360-system/370," Proc SJCC 1971, AFIPS, pp. 467–476.

[AB3] J. S. Gould, "On automatic testing of on-line real-time systems," Proc SJCC 1971, AFIPS, pp. 477–484.

[AB4] A. N. Higgins, "Error recovery through programming, Proc FJCC 1968, AFIPS, pp. 39–43.

[AB5] N. Mills, "NASDAQ—A user-driven, real-time transaction system," Proc SJCC 1972, AFIPS, pp. 1197–1206.

APPENDIX 2

Proposal for a Monitoring Register

Option I

Slowly changing signals, in millisec to seconds range, such as I/O channels busy, CPU busy, CPU in privileged state, etc.

Option II

Rapidly changing signals, internally in nanosec to microsecond range, but staticised for duration of following instruction. Include instruction address, opcode executed, requisters referenced, data reference address, success/failure for branches, byte/character/word length for variable length instructions, etc.

Option III

"Software filter option"—identifier(s) transmitted voluntarily by subject operating system.

Option IV

"Return identifier option"—destination for messages transmitted by external monitor to operating system.

```
          FORMATTED OUTPUT, LIVE FROM ADAM
DUMP OF ADAMS MEMORIES AT 12:25 FEB 07,'73,PROGRAM NAME =INHIBTIM15
SAMPLE NO. =    2,ADAM STATUS = D10A,PLUGBOARD READ DATA = 1420
-------------------------------------------------
    DISA   IN  OP  INHIBTIME
LOC BLE    HB
-------------------------------------------------
 00 3D37   03  FF          0
 01 3D37   03 [02]      1380
 02 D04F   03  69          0
 03 D053   03  25          0
 04 D054   03  68          0
 05 D056   03  31          0
 06 D057   03  69          0
 07 D058   03  47          0
 08 D059   03  20          0
 09 D05A   03  68          0
 10 D04E   03  31          0
 11 D04F   03  69          0
 12 D053   03  25          0
 13 D054   03  68          0
 14 D056   03  31          0
 15 D057   03  69          0
 16 D058   03  47          0
 17 D059   03  20          0
 18 D05A   03  68          0
 19 D04E   03  31          0
 20 D04F   03  69          0
 21 D053   03  25          0
 22 D054   03  68          0
 23 D056   03  31          0
 24 D057   03  69          0
 25 D058   03  47          0
 26 D059   03  20          0
 27 D05A   03  68          0
 28 D04E   03  31          0
 29 D04F   03  69          0
 30 D053   03  25          0
```

Figure 1. 'Provenance and Posteriority'

Identifier 02 signifies ALLOCAT module.

FORMATTED OUTPUT FROM HISTORY FILE WHO16JAN05
DUMP OF ADAMS MEMORIES AT 16:27 JAN 16,'73,PROGRAM NAME =WHO1
SAMPLE NO. = 1,ADAM STATUS = E03B,PLUGBOARD READ DATA = FF00

LOC	ADDRESS	COUNT	OVER WRIT	WRITES
00	4002	1	40	4
01	4004	1	41	1
02	7F37	2	43	1
03	7F38	30	43	1
04	7F5E	1	44	1
05	7F66	1	45	1
06	7F8E	1	46	2
07	0718	1	47	1
08	071B	1	48	1
09	0	0	49	1
10	0	0	4A	1
11	0	0	4B	1
12	0	0	4C	1
13	0	0	4D	1
14	0	0	4E	1
15	0	0	4F	1
16	0	0	50	1
17	0	0	51	1
18	0	0	52	1
19	0	0	53	1
20	0	0	56	1
21	0	0	57	1
22	0	0	58	1
23	0	0	59	1
24	0	0	5A	1
25	0	0	5B	1
26	0	0	5C	2
27	0	0	5D	1
28	0	0	5E	1
29	0	0	5F	1
30	0	0	60	2

Figure 2. 'Localization'
Sample period 12 hours overnight; observed write activity is the result of normal system initialization.

```
     FORMATTED OUTPUT FROM ADAM HISTORY FILE INH19FEB01
 SAMPLE NO.    6 TAKEN ON 13:14 FEB 19,'73 ADAM STATUS =D03B PLUGBOARD RE
PROGRAM INHIBTIM18 (D) INTERRUPT DISABLES (ONLY); (>100 USECS);COUNT & M


LONG DISABLES
  ALLOCAT+.24          194148
  DELTAGO+.19            1103
  COC CODE+.F2            874
  SSS+.111                833
  COC CODE+.480           697
  ECOCR2+.24              695
  SSS+.422                492
  MBS+.D3                 540
  T:GJOBS+.1              506
  T:RCE+.9                458
  T:RDERL+.1C             579
  GBG+.2D                 425
  CO:INO+.12              420
  CO:INO+.18              289
  COCOP56+.1D             310
  SSS+.157                  0
  T:OV+.2D                355
  COCSEND+.44             186
  COC CODE+.45C           332
  REG1                    330
  COC CODE+.467           325
  IOSCU+.1                266
  SERDEV+.15              208
  ALLOCAT+.63             282
  RBBAT+.44               279
  RE ENT+.1               272
  PULLALL+.15             252
  T:PULLE+.1           182598
  WAKEUP+.7               217
  T:CHS+.4B               184
  T:CHS+.1                184
  IOSST+.5                160
  T:DELUS+.4A             136
  SGCQ2+.1                  0
  SERDEV+.E               133
  STKOVF                  130
  COCOFF+.1               118
  NEWQ+.2F                 90
 OTHER FIELDS:-           160
```

Figure 3. 'Prolongation'
Cumulative maxima (microseconds) in one hour observation period.

FORMATTED OUTPUT FROM HISTORY FILE RES27JAN01
DUMP OF ADAMS MEMORIES AT 10:35 JAN 27,'73,PROGRAM NAME =RESP
SAMPLE NO. = 1,ADAM STATUS = D03B,PLUGBOARD READ DATA = 0000

LOC	INTERVAL	FREQUENCY	MAXIMUM
00	0	0	0
01	1	0	0
02	2	0	0
03	3	0	0
04	4	0	0
:	:	:	:
:	:	:	:
:	:	:	:
34	34	0	0
35	35	0	0
36	36	0	0
37	37	471	0
38	38	72	0
39	39	200	0
40	40	762	0
41	41	44	0
42	42	18	0
43	43	321	0
44	44	1771	0
45	45	19	0
46	46	10	0
47	47	3538	0
48	48	570	0
49	49	141	0
50	50	83	0
51	51	96	0
52	52	33	0
53	53	5	0
54	54	11	0
55	55	9	0
56	56	0	0
57	57	2	0
58	58	13	0
59	59	28	0
60	60	0	0
61	6645629	[122]	[68]

Figure 4. 'Distribution'
Variation in times to restore user environment.
122 observations exceed 60 microseconds.
Longest period = 68 microseconds.

FORMATTED OUTPUT, LIVE FROM ADAM

DUMP OF ADAMS MEMORIES AT 09:22 FEB 21,'73,PROGRAM NAME =BLKTIMES

SAMPLE NO. = 1,ADAM STATUS = D0AA,PLUGBOARD READ DATA = 0689

LOC	TRUE =REAL	BLOC KADR	FALSE =VIRTUAL
00	26804	3D80	203
01	168	3D70	110
02	2305	3D60	3274
03	0	3D50	78
04	6326	3D40	11011
05	15108	3D30	17748
06	0	3D20	0
07	0	3D10	0
08	0	3D00	0
09	432	3CF0	345
10	0	3CE0	32
11	0	3CD0	0
12	9	3CC0	19
13	151	3CB0	560
14	4613	3CA0	2744
15	11200	3C90	6068
16	3312	3C80	4468
17	5529	3C70	2772
18	11457	3C60	6766
19	0	3C50	33
20	148	3C40	0
21	415	3C30	0
22	0	3C20	0
23	61	3C14	4
24	20	3C00	0
25	10365	3BF0	529
26	49	3BE0	542
27	2290	3BD0	11
28	2709	3BC0	827
29	0	3BB0	2223
30	0	3BA0	10
31	97	3B90	223
32	4939	3B80	3607
33	1567	3B70	1515
34	0	3B60	22
35	0	3B50	124
36	0	3B40	167
37	0	3B30	26
38	0	3B20	711
39	0	3B10	2752
40	0	3B00	3237
41	30030	3AF0	852
42	2766	3AE0	682
43	10773	3AD0	0
44	15244	3AC0	465
45	8230	3AB0	0
46	15726	3AA0	9
47	13334	3A90	0

Figure 5. 'Utilization'

Times are cumulative microseconds of CPU execution in REAL or VIRTUAL mode during 10 minute observation period.

CHAPTER III

Tools and Techniques

A. MEASUREMENT

Tools and techniques for monitoring a computer system's performance have developed rapidly over the last five years. At the beginning of the period, analysts were faced with the question: How can I obtain any data on the system's performance? Today the question is often: Which tools should I select to measure which system characteristics? Although measurement technology has advanced dramatically, it (and associated areas) still require further development and, particularly, research.

Four papers on these topics are presented in this section. Warner reviews the historical development of monitoring tools and suggests where further monitor development will (or should) go. Schwetman's paper is concerned with the topics of designing experiments in which monitors are used and with the analysis of data from these monitors. He notes that performance variability is one of the difficult realities to address, and Bell's paper explains the magnitude, importance, and techniques to deal with this variability. These tools, techniques, and problems are highly technical; Jeffery's paper points out that this technology should be employed only if required.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, Computer Performance Evaluation;
Report of the 1973 NBS/ACM Workshop, March 27–30, 1973, San Diego, Calif. (Issued August 1975)

Measurement Tools

C. Dudley Warner

Tesdata Systems Corporation, Santa Clara, California 95050

Abstract

First generation computers were designed to operate in a serial fashion—performing one operation at a time (e.g., input, output, process). Performance evaluation was simply a matter of determining, with a watch or calendar, the time from the start of a job to the end. After several generations of computers, we now have systems with an enormous degree of complexity and parallelism.

In an effort to keep pace in the performance evaluation area, several computer manufacturers, as well as companies not directly involved in the manufacture of computers, have built a number of performance measurement tools. These tools cover the whole spectrum, from a simple device using electromechanical counters, to a system larger than most computers it would measure.

Future measurement systems will involve both hardware and software. Special software sometimes will communicate with the hardware monitor over a special I/O interface. These new measurement systems will then not only provide information about system performance, but will provide much more accurate job accounting information than is currently available, as well as doing a better job of scheduling.

Key words: Computer, evaluation, hardware monitors, measurement, performance, software monitors, throughput.

1. Introduction

This paper describes measurement devices used for evaluation of computer systems. It does not examine any measurement techniques. In an effort to develop a basis for "fruitful discussion," I have suggested the measurement tools that I think will evolve in the future. But first I must examine the origins of today's measurement devices and the current stage of development in order to lay the groundwork for the evolution that I envision. In tracing the history of these tools, I have not attempted to describe all that have been, or are, available. Instead, I have indicated only where advances or changes have occurred.

I have been the principal designer for much of the equipment that I discuss. Although it might be suggested that my selection for discussion is biased, the fact is that these are the ones that I know most about and am most competent to describe. They are also many of the landmarks in the changing measurement picture.

2. History

First generation computers were designed to process as fast as possible in two principal areas of application: scientific and commercial. The scientific processors were judged by how fast they could add, subtract, multiply, and divide; the commercial processors were judged by how fast they could manipulate data. These early processors were organized to operate serially—that is, they had to input the program and data before processing could begin. While processing, the computer could do no I/O. Upon the completion of processing, the information was output.

These machines were very easy to evaluate. You had to trigger a stopwatch when the load button was depressed, and you stopped the watch when the last line was printed.

You could improve execution time somewhat by streamlining the program, if you knew where to look, but processing improvement was limited by very slow I/O devices. Throughput evaluation of these systems meant using a watch or a calendar to time a program or a series of programs from beginning to end.

Computer manufacturers were quick to realize that this serial process was wasteful, so the next generation of computers was designed to perform the I/O functions in parallel with program processing. This presented a new set of problems to the marketing departments of computer manufacturers. Since a customer would evaluate several vendors' equipment by using benchmarks, it no longer was a simple matter of whose processor was the fastest. How well parallel I/O was handled became important.

In the case of IBM, the answer lay in measuring the degree of parallelism, or overlap, between processing and I/O. After several attempts to do this with software, a hardware device was designed. It was called (aptly and simply) "The Hardware Monitor."

This device was nothing more than a collection of standard instrumentation counters with some minor logic to interconnect them. The problem was that this rather simple device required a fairly elaborate interface to be designed into the computer.

Since the early computers had nothing like a wait state, this interface also had to have instruction recognition circuits for decoding the standard instruction loops used to wait for I/O completion. The Hardware Monitor counted instructions executed, channel 1 busy, channel 2 busy, unit record (equipment) busy, CPU wait, and total elapsed time.

Here are some of the hardware devices that evolved from the Hardware Monitor. All but one are for IBM machines.

Channel Analyzer—This device was functionally the same as the Hardware Monitor but much smaller, employing computer circuitry instead of the instrumentation counters. While the Hardware Monitor measured 6′ by 1½′ by 2′ in size, the Channel Analyzer was footlocker size.

Program Monitor—The Program Monitor performed the same function as a trace program but also monitored channel activity. It could trace the logic flow of a program running on the computer and record the results on magnetic tape. While a program might take from 10 to 100 times as long to execute while running in trace mode, the Program Monitor allowed the program to operate at full speed, because the Monitor was transparent to the host system. The drawback to widespread usage of the Program Monitor was the fact that the data reduction program could, and often did, take up to 7½ hours to reduce 5½ minutes of trace information to usable data.

"P" Snatcher—This device was similar to the Program Monitor. It used one Univac 1108 to trace another Univac 1108. It could capture only 17½ seconds of trace information at one shot, however.

Program Event Counter (PEC)—The PEC was an improved version of the Channel Analyzer with more counters. The two principal added features were a logic plugboard and comparators. The logic plugboard allowed for combining the interface signals in various ways prior to directing them to counters. The comparators provided the ability to compare storage addresses or storage content against up to six fixed quantities.

Execution Plotter—The Plotter was an attempt to overcome the data reduction burden imposed by the Program Monitor. It took the storage address, transformed it to a voltage through a D/A converter and displayed this voltage on the vertical axis of an oscilloscope. The horizontal axis was time; it was generated by moving film in a strip film recorder. In this way, core usage could be seen as a moving spot on film, where a bright spot meant high usage.

Systems Analysis Measuring Instrument (SAMI)—A larger version of the PEC, SAMI had more counters, more comparators, and more programmable logic. The unit was built in a System/360 Model 50 frame, contained 50% more logic than a standard Model 50, and consumed twice the power. One of the more interesting features of this machine was its relocation register which could be set by the host machine. The register allowed the comparators to operate on relocatable addresses. Although it worked from a measurement interface, the SAMI was the first device as far as I know that could also collect data using general purpose probes. This device also was the first to be used to monitor System/360.

Basic Counter Unit (BCU)—The BCU was the predecessor to all of the first commercially available hardware monitors—CPM I and II, SUM, Dynaprobe, CPU 7700 and 7800. It was to be used to service IBM customers and was not for sale. The reason given for not selling the BCU was that the Systems Engineer should receive and interpret the results in order to avoid erroneous conclusions that might be drawn if the customer were left to his own devices.

X-ray—X-RAY has all of the features of the first commercially available hardware monitors, with minor exceptions, and also some very useful improvements. The most notable is its ability to map core usage, which, in my view, is much more important than the simple trace technique employed in the Program Monitor and Execution Plotter.

3. Software Tools

While the hardware monitor has many features that make it ideal to do most of the measurement and evaluation jobs on computing systems, it lacks the ability to gather information at the individual program level in a multiprogram environment. The software monitor, on the other hand, lends itself well to this job. However, it is unable to gather accurate measurement information at the device level, and, of course, it imposes an overhead burden to the system it is monitoring.

Software monitors are designed to operate at three different levels. At the first level, they operate to give information about the total system, e.g., SUPERMON, Boole and Babbage's CUE. These focus on hardware resources and operating system functions which cause available wait time. The programs analyze the percentage of CPU wait according to the reason for the wait, I/O path availability, queue properties, and device accessibility. These programs most closely approximate the function of a hardware monitor.

The second type analyzes resource requirement at the individual program level, e.g., PROGLOOK, Boole and Babbage's PPE, Lambda Corporation's LEAP. These allow the user to see where his programs are spending their time and where program improvements can be made.

The last type is, basically, a set of job accounting routines, e.g., SMF. These show resource utilization such as CPU, direct access device, used versus requested core, and a completion code summary.

4. Where Are We Today?

The latest hardware monitors are beginning to incorporate minicomputers in their design. Since hardware monitors have, in the past, produced a large quantity of recorded data which is often quite overpowering, these new monitors provide some preprocessing. Or, as Hughes calls it, "software filtering."[1] [1]

Examples of these new devices are the Tesdata 1185, [2] and the COMRESS Dynaprobe 8000.[3] COMRESS claims that their system can measure job timing, task switching, queue delay timing, channel program activity, IOCS activity, etc. These are sizeable claims, but, at present, some subset of these can be achieved only by taking the measurement data produced by a monitor and the software data produced by the host system during a processing run, then merging both sets of data during a separate post-processing phase of the data reduction program.

At the other end of the price spectrum is the Micro-Sum,[2,4] a CRT device which attaches to the host system using probes. It can display 8 or 16 events as a percentage of elapsed time in histogram form. This device in its simplest form is very useful to the operations staff as a real time tool for resource balancing.

5. The Future: Some Thoughts

In the past, hardware and software monitors have been used on a once-in-awhile basis or by installations that have complex systems or more than one system or both. This limits a monitor's usefulness to those users that have large budgets. By developing additional uses for this monitoring equipment such as job accounting and resource scheduling, monitors can be made useful for all types of users or installations—large, small, complex, simple.

What I am suggesting is a process controller that is capable of accepting commands from the host system and that can interact with the operating system. These ideas are not new, and several successful attempts in this direction have been made already.[1, 5, 6]

5.1. Job Accounting

By maintaining job resource files in the process controller and allowing the controller to update each time an interrupt is handled, a great deal of the bur-

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

den and most of the overhead could be removed from the host system. If we couple this information (updated job resource files) with the hardware monitor measurement information, we can achieve a degree of accuracy and repeatability never before possible.

5.2. Scheduling in Real Time

From prior runs, operational characteristics and resource requirements can be catalogued by the process controller. Once it knows the requirements of the jobs currently in the job queue, the resource requirements of these jobs plus the request queues and the actual status of all I/O devices and data paths (measured information), the process controller can alter the sequence of these jobs to take advantage of underutilized resources.

5.3 Performance Measurement

Measurement information can evolve as a by-product of job accounting and scheduling rather than as primary, stand-alone data. It then serves as the long range planning tool for predicting when it will be necessary to add additional resources and/or upgrade the CPU.

Measured information is not an end in itself, but, rather, one of several inputs that will lead to an efficiently managed computing system. Current systems are too large and too fast to be scheduled by the human resource alone. What we must come to is a closed-loop system of process control.

6. References

[1] Hughes, J. and D. Cronshaw, "On using a Hardware Monitor as an Intelligent Peripheral."

[2] "Tesdata System 1000—Computer Performance Management System," Tesdata Systems Corporation, Chevy Chase, Md.

[3] COMRESS Dynaprobe 8000, COMRESS, Rockville, Md.

[4] Johnson, R., "Evaluation of the Micro-Sum Hardware Monitor," Computer Measurement and Evaluation Newsletter, Number 19, January 1973, pp. 19.36.

[5] Estrin, G. et al., "SNUPER COMPUTER; A computer in Instrumentation Automation," AFIPS Conference Proceedings, SJCC, Vol. 30, 1967, AFIPS Press, Montvale, N.J., pp. 645-656.

[6] Aschenbrenner, R., and N. Natarajan, "The NEUROTRON Monitor System," Digest of Fifth IEEE Computer Society Conference, IEEE 1971, pp. 31–37.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, Computer Performance Evaluation; Report of the 1973 NBS/ACM Workshop, March 27–30, 1973, San Diego, Calif. (Issued August 1975)

State of the Art: Experiment Design and Data Analysis

H. D. Schwetman

Purdue University, West Lafayette, Indiana 47906

Experimental observations form an important part of computer system performance evaluation. It is through experimentation that models are validated, simulations parameterized and systems tuned. This paper surveys several approaches to designing experiments to aid in the assessment of systems behavior. Data gathering tools and techniques are discussed, as are the important topics of data presentation and data analysis. The paper concludes with a critical examination of the state of the art of experimentation. The key problems are found to include: (1) a lack of generally applicable guidelines, (2) a missing link between low-level data and high-level questions and (3) a lack of means for dealing with variations in behavior attributable to variations in the workload.

Key words: Computer system performance evaluation; experimental assessment of system behavior; performance data analysis; performance data presentation; performance monitoring.

1. Introduction

Experiment design and data analysis as related to computer system performance evaluation covers a great diversity of topics ranging from the development and validation of analytical and simulation models to casual observation of a subsystem, such as a disk drive, in operation. In this paper, the term "experiment" will be restricted to mean a period of observation of a computer system in operation. The system being observed could be either an actual or a simulated system.

Experimentation is an important part of computer system performance evaluation. It is through experimentation that models are validated, simulations parameterized and systems tuned. In spite of its importance, design of computer-based experiments has received little attention in the literature. Experiments themselves have been described, but no base of knowledge and experience concerning experiment design is available.

Experiments on computer systems have produced many beneficial results, but there are limitations to an experimental approach to finding solutions to system performance problems. These limitations center on the variability of the system load found in normal operation. This problem, and several others, are discussed in this paper. It is hoped that as experience is gained, solutions to some of these problems will emerge.

An extensive bibliography covering performance evaluation appeared in a recent issue of Computing Reviews; this will be referred to as the CR Bibliography.[1] [1]

2. Classifying Experiments

An experimenter initiating a program of computer-based experiments has to make several choices about various techniques available to him. Many times he can choose between using an actual computer system or a simulated system; he can observe the system in normal operation or create a special test environment; and he can gather data at any one of several levels of detail. The purpose of the study, the tools and systems available, and the background of the experimenter all influence each of these decisions, and others, which must be made. This section presents features of experiments which can be used to classify them along the lines suggested by the choices open to the experimenter.

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

2.1 Computer System Used

One way of classifying an experiment is by the kind of system being used as the test bed. This discussion will consider experiments conducted on both actual operating computer systems and on simulated computer systems. Actual systems have some advantage over simulated systems because they already exist and because they are very realistic. Alternatively simulated systems may require a prodigious implementation effort, and the degree of attained realism is always questioned.

The major drawback to using actual systems is that questions about systems or features of systems which are not available cannot be answered, while a simulated system can be constructed to model almost any system or system feature. Also, there is the matter of cost; stand-alone usage of existing systems can be prohibitively expensive. However, it is not clear that using simulated systems is much less expensive. One simulated system is reported to consume five minutes of actual computer time to simulate one minute of time on the modeled system.[2] Trace-driven modeling is an interesting blend of these two types of experimental techniques.[3]

There have been several documented studies which use both kinds of systems. Many such studies are listed in the CR Bibliography.[1]

2.2. Operating Environment

Some experimenters may wish to compare the performance associated with several alternative configurations or combinations of system features. Such a comparison uncovers the fact that most performance indicators vary with both the system (hardware and software) configuration and with the job load. In order to make valid comparisons of performance, one must control the job load so that a given operating environment can be reproduced. This controlled environment has been achieved by using sets of benchmark jobs, synthetic jobs[4] and controlled job introduction techniques for batch systems[5] and terminal simulators for time-sharing systems.[6]

Another operating environment is found in the normal or production mode of operation. When observing the system in this environment, the experimenter has some assurance that he is gathering "typical" performance data, an assurance which may be lacking in the controlled environment. This normal (but uncontrolled) environment does present problems to the experimenter. For example, it may be difficult, if not impossible, to compare data gathered during separate periods of observation. Also, peculiar or unexpected values may be observed in the data, and there may be no convenient way of relating such observations to the job load being processed.

2.3. Purpose of Experiment

Another way of classifying experiments is by the expressed purpose. One series of experiments may be aimed at "tuning up" an existing system. Here the emphasis is on trying to answer questions such as: "Is the system operating at an "adequate" level of performance? If not, why not? If so, can this level of performance be achieved for less money?" Many "tune-up" studies have been conducted; most have produced at least some noticeable improvement (often because the system was operating so poorly).[7]

Another type of study may be directed toward evaluating different system configurations and different combinations of system options. One such study (unpublished) dealt with the desirability of acquiring a drum on a large System/360. The experiments included observing a system with and without the drum to determine whether or not the drum could be cost-justified (it was).

Some studies have been concerned with detecting and eliminating bugs within a system. Another study located system bottlenecks and was used to justify changes in job-processing priorities.[8] In summary, there can be several different reasons for conducting experiments, and the experiments can be classified by these purposes.

2.4. Level of Detail of Data

One other way of classifying experiments is by the level of detail of the collected data. In some situations, a stop watch, or perhaps some gross level job accounting information, may be all that is available or required. At another level, detailed job accounting data is used to produce information about resource utilization and the flow of jobs through a system.

Another level of data is characterized by sampling monitors (hardware or software). Here information is obtained about resource usage and even queueing

or contention for resources. At a very low level there may be some event-logging facility which can record (for later analysis) the stream of chosen events as they occur during system operation.

In attempting to compare these levels, one can generally say that the lower the level, the greater the amount of data and also the greater the cost of gathering the data. (Stopwatches are very cheap.)

3. Current Status—Data-Gathering Facilities

Successful analysis of computer systems using experimental techniques depends on access to good tools. When a simulated system is being used, the relevant tool is the simulator. When an actual system is being used, the relevant tools are encompassed in the data-gathering and analysis facilities. The state of simulation aids has been discussed in numerous other sources.[9] This section will concentrate on surveying the current state of data-gathering tools.

3.1. Job-Accounting Facilities

Most operating systems used with large third-generation systems come equipped with some job-resource accounting facility. Others have had such facilities added so that today most system users can use job-resource accounting data as a source of measurements. In some systems, this facility is quite extensive and potentially a source of useful data. For example, usage of many system resources such as CPU time, I/O facilities and disk space can be recorded for use as an evaluation aid. This data source can also provide information about the jobs flowing through the system and the system programs being used by these jobs.[10]

This type of data-gathering facility is of limited use in many experiments because the data does not reflect usage patterns; it only reflects total usage. For example, CPU idle time for an entire shift can be obtained, but CPU idle time for a particular minute cannot. Another problem occurs if the apportionment of use charges is not done in a systematic fashion. For example, in some systems an interrupted user program is charged for the CPU time required to process the interrupt, even if that program was not responsible for the interrupt. While this may be an equitable charging scheme, it does not enhance the value of job-accounting data as an aid to performance evaluation. Still other problems can be attributed to the coarse resolution of the system clock.

3.2. Sampling Monitors (Software and Hardware)

Sampling monitors sample and record the status of system variables. Hardware monitors record the status of variables at the circuit level. For example, they can record the usage of or contention for hardware resources,[11] but they are generally limited in their ability to record the status of resources or queues represented as tables in core. Because of this apparent limitation, information on the lengths of queues or the flow of jobs is not readily available to hardware monitors.

Software sampling monitors also record the status of variables, but at the software level. For example, they can collect information about the contention for resources and resource usage as seen at the operating system level.[12] Software monitors do impose an added load on the system. It is hoped that this added cost is outweighed by the benefits of getting the data. Also, there are some variables of interest which cannot be sampled by software monitors: memory interference is one such variable.

There are items of interest which are not readily available to any sampling monitor. For example, the interval between two events is not easily measured by sampling techniques. Also patterns or sequences of events are not usually obtainable.

3.3. Event Recording

A few operating systems have been equipped with a general event-recording facility.[13] This type of tool is probably the most general. It is capable of providing almost any information about the operation of the software system which could be desired by a researcher. It is also the most expensive in terms of effort required to implement and systems resources required by the data-gathering activity. The fine level of detail which is available is the cause of these problems. Also, the researcher faces formidable decisions in selecting items to be summarized in the report generation phase.

Some hardware monitors have been used as event-recording tools. This usually turns out to be an extremely difficult and time-consuming endeavor.

4. Current Status—Data Analysis

Display, reduction and analysis of the gathered data is a critical problem area in any experiment. It is especially critical in the area of computers because of the large quantity of data which can be gathered in a relatively short period of time. One experimenter using an event-record facility logged 1.6 million events per hour.[5]

The most primitive display of data consists of listing all of the events as they occurred. While this method of presentation may allow the experimenter to see sequences and patterns of events which may not show up in any higher level data display, it also may allow him to "not see the forest for the trees."

Data displays may consist of time-based bar graphs and frequency histograms or the more normal statistical quantities, such as the sample mean and standard deviation, the range, the min, the max, etc. These all represent reduced data which can be more easily interpreted. The problem is that each level of reduction involves a trade-off between the quantity of information and comprehensibility of the information.

A simple example follows. In one study, the CPU utilization was recorded by a hardware monitor. The resolution of the monitor was 5 seconds (i.e. the data was presented as a sequence of numbers, each representing the percentage of a 5-second interval the CPU was busy). When this data was summarized over a 1-hour period, the sample mean was observed to be 62 percent (pretty good). However, the sample variace was observed to be over 2000, which, based on other observations, was judged to be very high. Finally a time-based bar graph was produced; this graph showed that the data was a sequence of several one minute intervals of zero percent CPU busy followed by a sequence of intervals of 100 percent CPU busy. The user finally recalled that during the period in question the maintenance engineers had been using the system and were running intermittently some diagnostic programs. While perhaps an extreme situation, it does vividly illustrate problems associated with using "simple" statistics. It also vividly illustrates the continuing need for relating the observed data to the system job load.

5. Current Status—Experiment Design

In the field of statistics, a well-designed experiment produces valid results as efficiently as possible.[14] An experimenter observing a computer system has to produce a well designed experiment in this sense, but he also must solve some problems which are not normally found in experiments in other areas. Three features of computer-based experiments emerge as requiring special consideration: (1) the speed of data accumulation, (2) the dependency of observed results on the lob load and (3) a disparity between the level of questions asked and the level of data gathered. The status of the design of computer-based experiments can be placed as roughly equivalent to design of experiments in other fields, but with these additional factors, the design procedure may be more complex.

This section will not discuss experiment design from the statistical viewpoint. Instead it will focus on the three problem areas mentioned above. The discussion assumes that the importance of using reliable statistical techniques and of developing a plan prior to the beginning of experimentation is understood.

5.1. Speed of Data Accumulation

Events can occur at an extremely rapid rate in a computer system. This obvious fact can lead to problems when data is gathered from an operating computer system. Because of this speed, storage of the gathered data becomes a limiting factor as an experiment is being designed. Many times, the designer has to sacrifice detail in order to gather data over a significant period of time. Data is often compressed as it is gathered, at an increase in the cost of collection and perhaps at an additional cost of losing some of the desired information. Sampling techniques are sometimes used as a means of reducing the amount of collected data.

The speeds present also mean that event recording can be expensive in terms of resources required to gather the data. A software monitor can impose a significant load on the observed system. Hardware monitors also become more sophisticated, and hence more expensive, as greater resolution is required.

Because of the large amount of data which can be accumulated during the course of an experiment, the data analysis and data presentation phases assume an important role. It is impractical to view all of the data collected in many experiments. The experimenter has to develop statistics and data displays which accurately summarize and portray the factors being studied. There is a need for improvement in this area.

5.2. Dependency of Results on the Job Load

As stated earlier most performance indicators vary with both the load being processed and the configuration of the system. This often restricts the applicability of the results of an experiment to the environment in which it was conducted. Currently, there is no satisfactory method of characterizing a job load in terms of the demands it makes upon a system. This inability to characterize program and job behavior limits the use of performance evaluation techniques.[15]

In order to factor out the load as a contributor to variations in observed performance, the experimenter can choose between two techniques: he can conduct his experiments in a controlled environment, or he can observe the normal (uncontrolled) environment and hope that load-caused variations are averaged out over a long period of time. Neither technique is entirely satisfactory. Since a system typically functions in an uncontrolled environment, it is imperative that any results of an experiment be applied in this environment. Since there is no convenient way of characterizing the load, it is difficult to predict that the obtained results will be transportable to the normal production environment. There have been new systems which were selected and configured on the basis of sets of benchmark jobs and which did not function satisfactorily when installed.

When observing the system in normal operation, the experimenter is seeing the system in some part of its regular environment and has some assurance that beneficial results can be applied. However, in this environment, the experimenter is limited in his ability to compare results gathered across different periods of observation.

5.3. Disparity between Level of Questions and Supporting Data

Often, the questions posed at the onset of an experiment are of a subjective type at a rather high level of abstraction, while the data which can be collected is definitely objective and usually at an extremely low level. Examples of questions which may be posed follow:

1. Assuming only a limited expenditure of funds, what changes can be made which would result in increased job throughput?
2. What should be done to decrease system response time?
3. What will be the impact on system performance of adding an additional application (or more jobs) to a system load?

The data which can be gathered typically include items such as the percentage of time a data channel is busy or the amount of time the CPU is in wait state. There seems to be a missing link between macroscopic questions and microscopic answers. There is a definite need for a methodology which would guide experimenters in the selection of collectable data to support answers to typical questions.

6. Future Directions—Computer-Based Experimentation

What can we strive for in the area of computer-based experimentation? There seem to be three general areas of experimentation which should be improved. One is the training of experimenters to make better use of currently available tools and techniques. For example, modern applied statistics is being used routinely in other areas to solve problems which are similar to those faced by experimenters in performance evaluation. Knowledge of statistics can provide assistance in both the design of experiments and the analysis of data.[16]

Another aid to experimenters would be a collection of information about previous experimentation (both successful and unsuccessful) so that present efforts could benefit from previous efforts. Such a collection does not exist in the literature because 1) this type of experimentation is very specific to a particular operating environment thus usually not of wide enough interest to be considered for publication and 2) this type of activity often produces results which are considered proprietary and thus cannot be published (especially if negative results are produced).

Another area of experimentation which needs to be extended is in the area of developing a methodology or group of techniques which could ease the design effort. This could include development of models (analytical and simulation) which could be used as guides to experimentation. Such models could hopefully allow an experimenter to form questions at the previously mentioned higher or macroscopic level and then to determine those measurable quantities which influence the desired measure of performance. Also, some mod-

els could be used to predict performance in new operating environments.

The third area of improvement is concerned with describing or characterizing job loads. The present lack of load characterization methods limits the range of applicability of experimental results. The desired results in this area would include a method of presenting conclusions about present levels of performance as a function of the present load parameters. If this were done, then conclusions could be extended to modified loading conditions with some degree of certainty that the conclusions would remain valid.

7. Summary and Conclusions

This discussion has not detailed the design and analysis of successful computer-based experiments. Rather, it has presented 1) some background material, 2) an evaluation of the current state of experimentation, and 3) some suggestions for future directions. One of these directions could be development of a mechanism which would allow new experimenters to benefit from previous experiments. Currently, most experimental results have been isolated and fragmentary and have not been extendable to other types of computer systems or, in some cases, to other sites using the same system.

The status of tools available to experimenters is judged to be adequate but not perfect. Current problems center on lack of flexibility and on high costs (money and resources required) of some of the tools. However, the status of these tools is not significantly hindering successful experimentation. It is other factors which limit the experimenter. These other factors include:

1. A lack of techniques and experience upon which to build
2. A missing link between microscopic measurements and macroscopic questions
3. A lack of load characterization methods

An awareness of the problems in these three areas and attempts to solve them can lead to major improvements in the design and implementations of experiments in the area of computer performance evaluation.

8. References

[1] Anderson, H. and R. Sargent, "Modeling, Evaluation and Performance Measurements of Time-Sharing Computer Systems," Computing Reviews 13.12, (Dec. 1972), 603–608.

[2] Morganstein, S., J. Winograd, and R. Herman, "SIM/61: A Simulation Measurement Tool for Time-Shared Demand Paging Operating Systems," ACM/SIGOPS Workshop on System Performance Evaluation, (1971), 142–172.

[3] Sherman, S., F. Baskett, and J. Browne, "Trace-Driven Modeling and Analysis of CPU Scheduling in a multiprogramming System," Comm. ACM 15.12 (Dec. 1972), 1063–1069.

[4] Bucholz, W., "A Synthetic Job for Measuring System Performance," IBM Syst. J. 8.4 (1969), 309–31.

[5] Schwetman, H. and J. Browne, "An Experimental Study of Computer System Performance," Proc. ACM Conf. (1972), 693–703.

[6] Pullen, E. and D. Shuttee, "MUSE: A Tool for Testing and Debugging a Multi-Terminal Programming System," Proc. AFIPS SJCC (1968), 491–502.

[7] Bookman, P., B. Brotman, and K. Schmitt, "Use Measurement Engineering for Better System Performance," Computer Decisions (April 1972), 28–32.

[8] Stevens, D., "Overcoming High Priority Paralysis in Multiprogramming Systems, A Case History," Comm. ACM 11.8 (August 1968), 539–541.

[9] MacDougall, M., "Computer System Simulation, An Introduction," Computing Surveys 2.3 (Sept. 1970), 191–210.

[10] Stanley, W., "Measurement of System Operational Statistics," IBM Syst. J. 8.4 (1969), 299–308.

[11] Cockrum, J. and E. Crockett, "Interpreting the Results of a Hardware Systems Monitor," Proc. AFIPS SJCC (1971), 23–38.

[12] Holtwick, G., "Designing a Commercial Performance Measurement System," ACM/SIGOPS Workshop on System Performance Evaluation (1971), 29–58.

[13] Campbell, D. and W. Heffner, "Measurement and Analysis of Large Operating Systems During System Development," Proc. AFIPS SJCC (1968), 903–914.

[14] John, P., "Statistical Design and Analysis of Experiments," The MacMillan Company, 1971.

[15] Ferrari, D., "Workload Characterization and Selection in Computer Performance Measurement," Computer 5.4 (July/Aug. 1972), 18–24.

[16] Freiberger, W. (ed.), "Statistical Computer Performance Evaluation," Academic Press, 1972.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406 Computer Performance Evaluation: Reopt of the 1973 NBS/ACM Workshop, March 27–30, 1973, San Diego, Calif. (Issued August 1975)

Computer Performance Variability

Thomas E. Bell

The Rand Corporation,* Santa Monica, California 90406

The performance of a computer varies significantly, even when it is subjected to the same load. Analysts who are performing between-machine comparisons, predicting performance, or merely trying to understand performance can be led to incorrect decisions if random variability is interpreted as representing real differences. Tightly contolled tests employing a flexible synthetic job indicated that elapsed time, processor time, and response time vary enough to deceive analysts. Several trends seem to indicate that variability will increase with time, so the effect will increase in importance. Both computer manufacturers and performance analysts should take specific actions to preclude problems due to computer performance variability.

Key words: Computer performance analysis, measurement, performance monitoring, performance variability.

"It ran 20 percent faster when I cut out the I/O buffering."

"My simulation is accurate to within 10 percent every time."

"This system was chosen because it did 14 percent more work than the next best system."

Statements like these nearly always assume that the value for a performance metric is a single, fixed number; if a test is run twice—or ten times—the result will be the same in every case. This assumption—stated more succinctly—is that there exists zero within-sample variability. Therefore, the variation between samples (e.g., several runs on the same computer) can be disregarded in comparison with the variation between samples (e.g., runs on computer A compared with runs on computer B). If the standard deviation (a measure of variability) of run times on computers A and B were always far smaller than the difference in run times on the two machines, the within-sample variability would clearly not be significant. Unfortunately, this nice situation does not always obtain, and the analyst is presented with difficulties in many analyses.

*Now located at TRW Systems Group, Redondo Beach, Calif. 90278.

1. Importance of Variability

Most analyses of computer performance involve the comparison of one situation with another. The two situations might be the desired processing capacity and the predicted capacity, or the capacity of one system and that of another system, or the capacity of the system before a change and the capacity after the change. In these cases variability within a sample obviously will make the analysis more difficult and expensive. If the confidence in a difference needs to be increased, more samples need to be taken in order to obtain better estimates of the population mean. This process is often so expensive that analysts settle for a single sample of each population and assume that no variability exists; the single sample is assumed to accurately represent the population. If variability is large and differences are small, the analyst may choose the wrong conclusion.

A different situation also can lead to problems. If an analyst takes a single sample of one metric of performance (e.g., throughput) and attempts to relate it to a single sample of a load metric (e.g., CPU utilization), he may conclude that a relationship holds true

when it is false. An example of the danger in this procedure is plotting a graph of throughput versus CPU utilization using a single sample of each variable for each degree of loading. If variability does exist in the performance of the computer, the graph will exhibit a wavy line even if a smooth line would represent population means. The analyst then would be led to investigate the causes of the false "wiggles" rather than concentrating on the system's real characteristics.

2. Magnitude of Problem

The suggested situations above would be unworthy of consideration if computer performance variability were small; some indication of the magnitude of the problem is necessary to evaluate its importance. Several people at Rand and elsewhere have used a modification of the Buchholz synthetic test job to determine the magnitude of variability in strictly controlled test situations on IBM and Honeywell equipment. This test job (as modified) includes embedded measurement in the form of interrogations of the system's hardware clock and recording of the elapsed time between certain major points in the program execution. The job is structured as follows:

1. Set up for the job's execution,
2. Set up for running a set of identical passes with a prescribed I/O-CPU mix,
3. Execute the set of identical passes and record in memory the time of each pass's start and finish.
4. Print the resultant execution times and compute some simple statistics for them,
5. If requested, return to step 2 to repeat the operations for a new I/O-CPU mix,
6. Terminate the job.

The measurement of each pass provides a number of identical samples from which the analyst can determine the variability within a job. By running the job in a number of different environments (e.g., concurrently with another, similar job), a known situation can be created to determine variability between jobs. Other statistics such as CPU time, I/O counts, and core used can also be employed.

The simplest situation to investigate is one in which no jobs other than the test job are active. The system is initialized and no other activity is initiated except for the single job. In this simple test the parameterized test job is set to run in either of two modes—as a totally CPU-bound job or as a totally I/O-bound job.

The elapsed time to execute the CPU-bound portion of this job on Rand's 360/65 evidenced no variability within the job that was above the measurement's resolution. The I/O-bound portion's elapsed time, however, typically varied enough to result in standard deviations (for 20 identical executions) of about .5 percent of the mean (mean of about 35 seconds). This small value indicates that, at least within a stand-alone job, variability is not impressive.

The situation becomes less encouraging when the comparisons are between separately initiated jobs. The statistics of interest now include initiation time, termination time, and average execution time of a pass through the timed loop for each separately-initiated job. The initiation time of the jobs (accounting time-on until control passed to the job) averaged 8.12 seconds with a standard deviation of 5.03 percent of the mean. The termination time (job execution completion until accounting time-off) averaged 3.72 seconds with standard deviation of 16.7 percent of the mean. The sample size of identically run jobs is too small for computation of meaningful measures of variability; in two specific instances, however, the average elapsed time to execute the CPU portion differed by less than the resolution of measurement. The average execution time for the I/O-bound portion, in two instances of samples of two, changed by 0.4 percent and 1.2 percent. (Allocation of files was done identically in each instance.) Although the within-sample variability appears small for internal portions of a job, the initiation and the termination times vary significantly. Elapsed time may be inadequate for evaluating performance changes in short jobs.

Thus far, elapsed time has been the only statistic used for evaluation of variability. Another statistic, I/O counts (EXCP's on a System 360 operating MVT), have proven to have virtually no variability. Other statistics of interest exist, and one of the most commonly used is CPU time. While elapsed time is an easily defined and obtained indicator of job performance, CPU time is often complex. Even its definition is open to dispute; major parts of a job's CPU activity may not be logically associated with that job alone. For instance, the rate of executing instructions may be reduced as a result of activity of a channel or another processor in the system. In addition to the definitional problem, accounting systems are often implemented in ways that users feel are illogical. These problems are seldom of importance when a job runs alone in the system, but may be critical during multiprogramming

or multiprocessing. Repeated runs of our test job on the 360/65 did not indicate any meaningful variability when the CPU-bound job was run stand-alone, but the I/O bound job, in two cases, provided CPU times of 29.4 seconds and 28.7 seconds (a difference of 2.4 percent of the mean). The I/O variability, however, does not appear critical because this job ran, stand-alone, for an elapsed time of approximately 780 seconds; the difference of 0.7 seconds is therefore less than 0.1 percent of the elapsed time. Under multiprogramming results vary more.

Although the CPU charges should not vary when the job is run under multiprogrammed rather than stand-alone, our results indicated that the results contained both a biasing element and a random element. The charges for the CPU-bound job went up from 583 seconds (stand-alone) to 612 seconds (when run with the I/O-bound job) to 637 seconds (when run with a job causing timer interrupts every 16.7 milliseconds) to 673 seconds (when multiprogrammed with both the other jobs). The changes in CPU charges are clearly dependent of the number of interrupts the system handles for other jobs on the system. The largest CPU charge observed in this series of tests with the CPU-bound job was 15 percent over the stand-alone charge, but larger biases can be obtained by running more interrupt-causing jobs simultaneously. In one particularly annoying case, the author observed a job whose CPU charges differed between two runs by an amount equal to the smaller of the two charges—a 100 percent variation! I/O-bound jobs often experience CPU charge variability of equal relative magnitude, and users with I/O-bound jobs have come to expect 30 percent variations in their charges. These problems are not unique to IBM equipment. We found virtually the same sorts of variability when running on a Honeywell 615 dual processor. Variability in the charges for jobs in other non-paged systems appears to be similar.

The measures discussed above are primarily oriented to batch processing; slightly different measures are usually meaningful for on-line systems. In the case of on-line services, response time is usually the measure of most interest. An on-line system operating with low priority in a computer would be expected to have variable response, but relatively constant response is usually expected when the on-line system is given a priority only a little below the operating system itself. We ran a series of carefully designed tests to provide an indication of this assumption's validity on the WYLBUR text editor in Rand's normal environment. A heavily I/O-bound activity (listing a file on a video terminal) experienced response time with a standard deviation 23.2 percent of the mean. A heavily CPU-bound activity (automatically changing selected characters) had reduced variability—15.6 percent of its mean. A variety of other editing functions experienced similar variability under a pair of different configurations. The standard deviations as percentages of means ranged from a low of 5.7 percent, to a typical 12 percent, to a high of 30.8 percent. These values were obtained with different levels of user activity on WYLBUR by real users (as opposed to our artificial load for testing), but did not include the variability often introduced by time-sharing systems since the 360/65 is not time-shared.

Our standard synthetic job was executed on a Honeywell 615 system under its time-sharing system. The elapsed time to execute each pass through the CPU-bound portion was recorded to provide an indication of the variability of response time during normal operation on that system. Each pass could be executed in about 9 seconds, but on occasion the average was as large as 597 seconds with the standard deviation exceeding the average. (It was 598 seconds.) Variability of this magnitude can produce highly anti-social behavior by users and ulcers for analysts.

3. Future Trends

Computer performance variability will probably not get worse before it gets better. It will probably just keep getting worse. Reducing variability (and retaining meaningfulness) appears to be an objective that is inconsistent with improving functions and/or performance. Increased functions can be acquired at the expense of performance, or they can be included in a system with little performance degradation by employing performance enhancements in the new functions' designs.

Many performance enhancements (either for the sake of new features or for performance's sake itself) inherently introduce variability. For example, rotational position sensing on disk drives can improve average response time and average throughput. However, it introduces another randomizing element in computer operations and therefore increases variability. Making resources available to more independent jobs will increase variability as each job adds its bit to the variability of all the others.

Vendors appear to be feeling the pressure of user demands for improved performance, and will probably respond to this pressure by improving performance. Many of the techniques used to improve performance will increase the magnitude of variability in job and system performance as overhead operations become more sophisticated and new hardware sometimes reduces parts of processing time.

4. Any Hope?

The presence of large variability in peformance measures makes performance quantification and improvement difficult. Four actions may reduce the variability-introduced difficulty of these tasks.

4.1. Improve Measures

Vendors' systems are often implemented in ways that make the variability larger without any apparent offsetting advantage. Performance analysts should not need to contend with artificially-introduced variability in addition to the naturally-existing variability in actual system performance. For example, the handling of I/O interrupt processing time in some systems is particularly bad; the processing is merely charged to the job in control at the time the interrupt occurs. This could be changed easily, but the vendors have not demonstrated an eagerness to correct the problem.

4.2. Use Better Statistical Techniques

Performance analysts should not depend on a single value from a population to represent that population when it possesses unknown variability. Performance analysts should employ, at least, the rudimentary statistical tools taught in beginning statistics courses.

4.3 Quantify Variability

Individuals should share information about variability so that its magnitude will be known during the formulation of a performance analysis plan. Better tests need to be designed for this quantification than the rudimentary ones in this discussion.

4.4. Design Hypotheses Better

Much of performance analysis consists of testing hypotheses. Some measures clearly have larger variability than others; hypotheses should be designed to employ the ones with least variability when several alternatives exist. In addition, the hypotheses should be worded so that testing can be performed in a situation introducing the least variability that is practical.

These actions are so simple to implement that the severity of problems posed by variability can be reduced when necessary. There is good cause for hope that we can deal with computer performance variability, but wishing for an end to variability is futile.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION; REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Domains For Performance Measurement

S. Jeffery

National Bureau of Standards, Washington, D.C. 20234

In lieu of an integrated approach to performance, it may be helpful to propose a structure for consideration of the entities of Performance Measurement: systems, applications, and measurement techniques. It is proposed that these entities can be compartmentalized into "domains," for the categorization of performance measurement. It is likely that definite domains will be uncovered indicating the use of performance measurement, or more important and less widely recognized, where it is not cost-effective to perform system measurement.

The tools for measuring performance—hardware monitors, software monitors, and accounting systems—are discussed in terms of system level or application level management programs.

Several tasks are suggested that need to be addressed: (1) the gathering of currently available information on the use of accounting systems, and the development and publication of guidelines for the employment of accounting data; (2) the development and postulation of a set of Computer Performance Evaluation domains; (3) a Performance Measurement Handbook comprising guidelines for utilization of computer performance evaluation over all domains.

Key words: Accounting systems; hardware monitors; performance evaluation; performance measurement; software monitors.

1. Increasing Interest in Performance Measurement

A barometer of interest in a particular area of the computer industry is the time devoted to the subject at the AFIPS Joint Computer Conferences. Performance measurement's importance was indicated by the allocation of no less than six sessions at the 1972 Fall Joint Computer Conference. This indicator of merit, along with the amount of space allocated by the technical periodicals, and the many technical conferences devoted to the subject (this one included), has caused a bandwagon effect advocating, if not encouraging, the use of performance measurement tools and techniques. But this indicator may be misleading. A survey, taken at this writing, revealed fewer than 200 hardware monitors in use (by some 1000 installations). It also revealed only about 1500 software monitors in actual use. If we are to believe these figures, apparently less than 10 percent of our medium and large scale system installations are applying software or hardware monitors to performance measurement. Perhaps a lack of knowledge about the proper role and use of performance measurement, and specifically, about the devices and techniques has made management reluctant to apply these and other aids.

Possible management questions include:

- What are meaningful measurements?
- When is the appropriate time to measure?
- What are the criteria for the selection of these tools?
- Are there problem-specific tools?
- Can performance measurement lead to improved reliability or to a productivity index for management?

2. A Structure For Evaluation of Performance Measurement

Since answers to these questions are not readily forthcoming, tied neatly in a package, some selective

efforts must be made to provide information which will instill the confidence in management to invest in and undertake performance evaluation.

In lieu of an integrated approach to performance measurement, it may be helpful to propose a structure for consideration of the entities of performance measurement: systems, applications, and measurement techniques. Except in cases of primitive sequential batch machinery, consideration of any of these entities alone is insufficient. It is proposed that these entities can be compartmentalized into "domains," for the categorization of performance measurement. It is likely that there will be uncovered definite domains that indicate the use of performance measurement, or even more important and less widely recognized, where it is not cost-effective to perform system measurement.

3. The Tools—Measurement versus Prediction

Before a discussion is presented on the applicability of specific tools to specific problems, a classification of the tools should be made. The tools used to measure performance include hardware monitors, software monitors, and accounting systems; conspicuously absent are simulation and benchmarking.

There is a spectrum of tools and methods which spans from measurement to evaluation with many combinations in between. Monitors are at the measurement end; the analysis of their output by man or program is a movement toward the evaluation end of this spectrum. Simulation, by language or by package, is a predictive tool rather than a measurement tool and is closer to the evaluation end of the spectrum. Experiments have been carried out with a simulator being driven by on-line measurements from a monitor. This is one way a simulation can be calibrated or validated to increase its reliability as a predictor, but the simulation is not itself a measurement tool. Similarly, simulation models have been derived directly from accounting data.

Benchmarking is also often considered a tool for performance measurement. The benchmark itself, however, serves as a workload for timing in computer system selection. It is in the area of computer or system selection that measurements obtained during a benchmark run can be used to determine performance. Another use of benchmarking is in comparison of timings before and after a system is "tuned." The percentage difference in the timings indicates relative improvement. It should be remembered, however, that a benchmark is only a partial representation of the complete workload. The benchmark is only valid as a predictive tool when this subset accurately reflects the characteristics of the workload.

4. Application Programming Performance Measurement and System Level Performance Measurement

In discussing performance measurement, one should first ask whether it is the system level or the application level programs which should be measured. While performance may be improved in both areas, the proper application of the tools is dependent upon this first cut at measurement goals.

4.1. Application Programming Performance Measurement Tools

The improvement of performance for an application program has traditionally been based upon reducing the run time within a serial processing environment. (It should be noted that the individual reduction of run times for each program of a set processed in a serial processing environment does not necessarily mean a reduction in the total run time of the set within a multi-programming environment.) The running time for such a program may be reduced by two methods: (1) more efficient program coding, or (2) more efficient file organization.

4.1.1. Software Monitors

The program-oriented software monitors are of either the system dependent, sampling type or the instrumented program type. In the first type the software monitor is run as an application program within the computer system. An interrupt triggered by a timer causes control to be relinquished to the software monitor which takes a "snapshot" of various registers and records them on tape. By this sampling technique the relative activity within various areas of memory can be displayed and compared with a program listing.

In the second type, the source program is instrumented to collect data on itself. This can be done by the programmer, by a preprocessor, or by a compiler.

Such systems are of necessity language dependent but are often useful for debugging as well as performance measurement.

4.1.2. Accounting System Information

Many accounting systems collect comprehensive, detailed information about program activity. Descriptive data (blocking, number of records read, file open time, overlay calls, etc.) enable the programmer to obtain gross data about program performance and extensive data about file usage.

4.1.3 Hardware Monitors

The first hardware monitors could not easily identify specific areas of high activity within main memory. They had only two address comparators and no ability to correlate program identification with memory activity. Newer hardware with more comparators and many accumulators provided an easier method of locating main memory areas of high usage. This same address content increment logic, adapted from minicomputer memories, permitted measurement of instruction activity by attaching probe points to the instruction register, thus giving the programmer a picture of the instruction activity which could presumably enable him to substitute faster instruction sequences for slower ones. The minicomputer-equipped hardware monitors easily permit memory activity mapping for optimization. The latest announced features include up to 20,000 memory counters to collect information such as channel time, device time, seek time, and interrupt time, for each program processed in the system. Such information is obtained from a set of probes and time stamped as it is collected in the minicomputer's memory.

4.1.4. File and Program Organization

Any of the above methods can provide information about the spatial and temporal pattern of reference to main and secondary memory. In a paged environment this information has taken on increased importance in light of Denning's working set model. A related question is the organization of files in a storage device with location dependent access time. Any of the measurement methods which record by channel and device will be valuable here.

4.2. System Oriented Performance Measurement Tools

The improvement in performance of a total system can create a reduction in total system processing time and the potential for cost savings. It must be emphasized that these savings can only be realized if there is a real increase in useful computing or the real reduction of fixed expenses. For example, if the upgrading of a configuration to accommodate an additional workload can be delayed by improving the overall system performance, then the additional workload can be placed on the original machine. Improved system performance may also eliminate the need to staff and pay for a second or third shift of computer operations. System performance improvement may also reduce response time and program turnaround time. Software monitors, hardware monitors and accounting systems can all be used to aid in system performance improvement.

4.2.1. Software Monitors

Software monitors used for system performance add an overhead which can distort the performance profile. A software monitor can examine system queues, CPU utilization (including software monitor overhead to correct for the monitor-introduced distortion), scheduling and resource conflicts, channel balance, channel utilization, disk read/write head movements, instruction activity, etc., but cannot effectively monitor operating system functions because of the program lockouts in supervisory state and the masking of interrupts. The software monitor permits examination of core usage, queue lengths, and instruction activity with relative ease. Because the software monitor relies on the host computer's system timer, the software monitor does not produce an independent measurement as does a hardware monitor. System software monitors are system dependent in the same way as the first type of application monitor is.

4.2.2. Hardware Monitors

The job for which the capabilities of the hardware monitor are most closely matched is measuring such system parameters as CPU utilization, channel balance and utilization, device delays, etc. Advantages include relative independence of hardware and system being

measured and no overhead to distort the system being measured. Against this, there are many system functions for which there is no obvious correlation measurable by a hardware monitor.

4.2.3. Accounting Systems

Systems accounting is essentially instrumentation in the system software for self-measurement. It is analogous to the second type of software monitor for application programming. Typically the information collected is about system status at the start of a system event (input/output, schedule, queue, etc.), conclusion of a system event (program termination), or at a specified interval of time. These events are time stamped and written to magnetic tape or disk. Analyzer programs can be written or obtained to sort this information by time of day, program, type of program (COBOL compilation, FORTRAN compilation, COBOL execution, etc.) and provide information such as memory utilization, blocking factors, mix of program types, etc. This accounting information is easily collected, requires no additional purchase of software or hardware, and provides an excellent first profile of system performance. There is, of course, some overhead associated with the data collection as there is with a specific software monitor. The more sophisticated accounting systems should be considered as built-in software monitors of the first type.

5. Is Performance Measurement Always Needed?

Performance measurement costs money. In most cases thousands of dollars are required for a meaningful measurement effort. It is not always clear that the improvement in performance justifies the costs. The installation manager should examine the needs of the installation, the tools, the costs of using these tools (equipment as well as personnel costs), and the possible benefits before making any decision. For instance, the first and obvious tool for measuring system performance is usually supplied with the system—the accounting package. This certainly is indicated as an initial Computer Performance Evaluation (CPE) device. By further example, unless the installation possesses an unusually large pool of measurement resources, one should not monitor for the sake of monitoring. One should have a definite reason which can usually be stated in the form of a problem. Examples of such problems are:

(1) saturation of the system,
(2) poor turnaround (for batch systems)
(3) poor response time (for time-sharing systems)
(4) excessive cost of suggested vendor hardware augmentation
(5) customer complaint about variability of billing data

This set of problems (and others not included) require solutions; these problems sooner or later will cost the installation money. The manager must decide how to solve the problems in a manner consistent with organizational goals.

6. Some Short-Range Solutions

As mentioned above, accounting systems can provide considerable measurement of performance. They usually can be obtained at minimal or no cost from main-frame vendors. However, little information is currently available on the effective use of this context. It would seem that an important task to be addressed would be the gathering of such information and the development and publication of guidelines for the employment of accounting data by facility managers in performance measurement. Using this immediately usable body of knowledge as a base, there would be developed a set of desirable accounting data features which would influence the design of future systems.

Another area which appears to have near-future attainability and usability would be the development and postulation of a set of CPE domains. This would enable not only the organization of future, advanced CPE work, but would provide a structuring for aggregation of data reflecting the appropriateness of CPE. Finally, having developed a structure for organizing techniques and encoding data, the necessary prerequisite has been established for undertaking a development of a long-range nature: a performance measurement handbook, comprising guidelines for utilization of CPE over all domains.

7. Bibliography

Advanced Management Research, Computer System Performance Measurement, New York, 1972.

Allied Computer Technology, CPM II, Computer Performance Monitor System Summary, Santa Monica, Calif., 1969.

Bard, Y., "Performance Criteria and Measurement for a Time-Sharing System," *IBM Systems Journal*, 10:3, (1971), p. 193–215.

Bell, T. E., Boehm, B. W., and Watson, R. A., "Computer System Performance Improvement: Framework and Initial Phases," *The RAND Corporation*, R–549–PR, August 1971.

Bonner, A. J., "Using System Monitor Output to Improve Performance," *IBM System Journal*, 8:4 (1969), p. 290–298.

Comress Corporation, Dynaprobe, Dynapar—Computer Performance Measurement, Rockville, Md. 1972.

Drummond, M. E., Jr., "A Perspective on System Performance Evaluation," *IBM Systems Journal*, 8:4 (1969), p. 252–263.

Morrison, R. L., "An Approach to Performance Measurement," *IBM Technical Report*, TR00.2272, February 15, 1972.

Murphy, R. W., "The System Logic and Usage Recorder," AFIPS Fall Joint Computer Conference, Vol. 35, (1969), p. 219–229.

"Savings from Performance Monitoring," *EDP Analyzer*, Vol. 10:9 (September 1972).

Shemer, J. E., "Computer System Instrumentation and Performance Measurement," *Computer*, (July–August 1972), p. 17–48.

Tesdata Systems Corporation, X-Ray Computer Performance Measurement System Reference Manual, TM–105–2, Chevy Chase, Md. 1972.

U.S. Air Force Logistics Command, "A Survey of Computer Performance Monitors and their Potential Applications in the Air Force Logistics Command," *Data System Simulation Report DSS Project 01–70*, March 1971.

U.S. General Accounting Office, "Opportunity for Greater Efficiency and Savings Through the Use of Evaluation Techniques in the Federal Government's Computer Operations," B–115369, *Report to Congress*, August 22, 1972.

B. THEORY AND MODELS

Analytical models may be used for very immediate requirements to understand or predict the performance of a computer system. Alternatively, they can be used to describe a theory of computer system performance. The papers in this section deal with a variety of approaches to both types of models and to the general topic of theory development. They provide an overview of the strength and limitations of each approach as well as describing each approach's primary characteristics.

Denning's and Muntz's paper discusses one of the best-known approaches to analytical modeling—queuing theory. The paper by Kimbleton discusses the capabilities and limitations of analytic modeling for computer system sizing and tuning. Johnson's paper deals with an entirely different approach; he employs Petri nets and probabilistic transitions to determine steady-state conditions. The approach described in Hellerman's paper employs the concept of the computational work performed in a single step. Kolence's paper suggests that the concept of work be pursued, but through analogy with the concept of work as defined in Physics.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION;
REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Queueing Theoretic Models

P. J. Denning and R. R. Muntz

University of California, Los Angeles, Los Angeles, California 90024

Tradeoffs between methods of solving analytical queueing models are discussed. It is suggested that representing the multiple (simply defined) resources of a computer system and the sequencing of tasks among these resources gives models which are simple enough to yield to analysis and yet are applicable to systems of interest. Theoretical results from the study of such networks are summarized and directions of future research are briefly discussed.

Key words: Analytical models, evaluation, measurement, networks, performance, queues.

As with other performance evaluation methods, analytic studies of computer systems yield only approximations to the solutions we desire. In analytic modeling several distinct types of compromises can be made:

1. In the abstraction of the real world to a mathematical model. Examples here include the assumptions of certain arrival processes, service time distributions and idealized service disciplines.
2. In an approximate analysis of the mathematical model. The diffusion approximation is an example of an approximate solution technique.
3. In settling for a less complete solution, e.g., mean response time rather than the distribution of response time, or equilibrium distributions rather than transient solutions.
4. In settling for a less convenient form of solution, e.g., a Laplace transform or a computational procedure is obtained rather than an explicit solution.

It is usually possible to increase preciseness in one area by introducing additional compromises somewhere else. Unfortunately much of the work in analytic modeling has concentrated on exact solutions for models which are gross simplifications, i.e., the compromises are made in constructing a mathematically tractable model. Too little attention has been paid to approximations and bounds on performance. However, it should be pointed out that the real question is whether the analysis yields results which are valid in a practical sense and not whether the model appears "over simplified." For example Lassettre and Scherr [1] [1] used a simple machine repairman model for TSO with considerable success; although without their experimental verification of its applicability it is likely that the model would be dismissed as "over simplified." Where to compromise is an exceedingly difficult question and can really only be determined by experience in applying analysis.

We are faced with the problem of determining what characteristics of a system must be included in a model to get reasonably accurate results. The "classical" queueing model for computer system scheduling [Figure 1] has assumed a single resource. The results of analysis of this type of model often do not agree terribly well with observed behavior. More recent research has been directed toward representing the multiple resources of a computer system and the sequencing of tasks among these resources. As one might expect this generalization to multiple resources has required that the characterization of the use of individual resources be less precisely defined. The indications thus far are that this tradeoff gives models which are simple enough to yield to some analysis and yet are applicable to systems of interest [2,3].

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

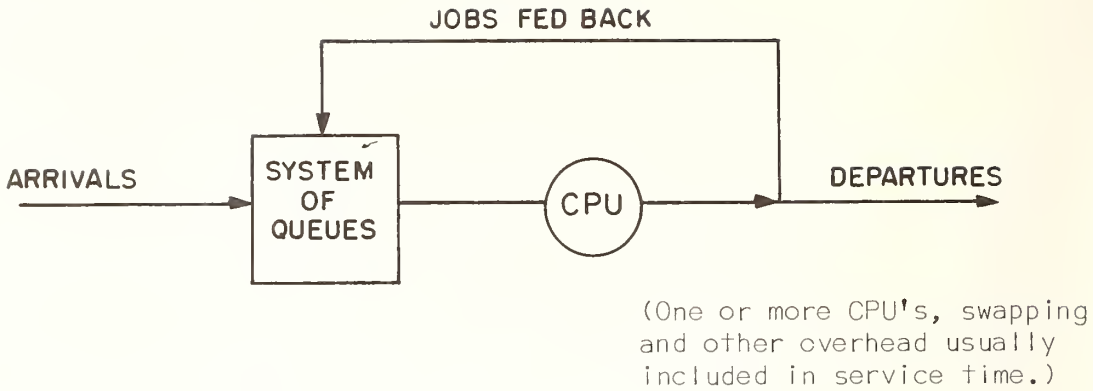


Figure 1.

1. Queueing Networks

A queueing network consists of a set of service centers $\{S_j\}$. Each service center comprises a queue or queues, one or more units of some given resource, and a service discipline. Each class of customer in the network (customer = job = task) has a transition matrix $p^{(i)} = [p_{jk}^{(i)}]$, where $p_{jk}^{(i)}$ gives the probability of a customer in class i requesting service at S_k after receiving service at S_j. Each class i of customer has a set of service distributions $\{F_j^{(i)}\}$ where $F_j^{(i)}(x)$ gives the probability that the service time of a class i customer at S_j does not exceed x. The state of an m-service-center system is a vector $(x_1, \ldots, x_m)$ in which x_j represents the state of S. The theory of networks has yielded the following results.

1. One can solve for the state-probabilities of an arbitrary network if each S_j is one of these four types:
   a. the discipline in S_j is processor sharing, $F_j^{(i)}$ arbitrary for all i,
   b. S_j contains enough units of resource to avoid queueing, $F_j^{(i)}$ is arbitrary for all i,
   c. the discipline in S_j is *FIFO*, $F_j^{(i)}$ is exponential for all i.
   d. the discipline in S_j is *LIFO*, $F_j^{(i)}$ is arbitrary for all i.
2. Servers of the four types can be mixed in any network.
3. Customers may change class dynamically. (Useful for modeling customers changing from one mode of behavior to another.)
4. The network can be open (number of customers in it variable) or closed (number of customers in it fixed).
5. Various types of state dependent service rates can be modeled. (This is useful for example, in approximating the behavior of secondary storage devices which operate more efficiently under higher load.)

The above work is an outgrowth of queueing network theory begun by Jackson, Gordon, and Newell; the latest results quoted above came from F. Baskett at Stanford, R. R. Muntz at UCLA, K. M. Chandy and F. Palacios at the University of Texas at Austin.

A special case, the central server network [7] has been studied extensively since it represents a common system configuration:

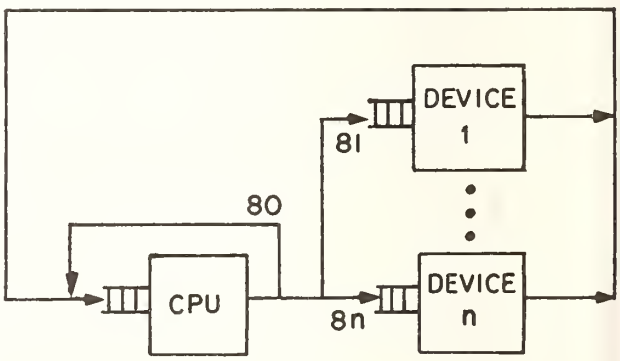


Figure 2. Central Server Model

The transition probabilities of the above network are $q_0, \ldots, q_n$. Experimental work at the University of

Texas at Austin (by F. Baskett) and the University of Michigan (by C. Moore, S. Kimbleton, and B. Arden) showed that this and similar types of networks have the following property: when the original service distributions are replaced by exponentials having the same means as the originals, the mean queue lengths and server idle probabilities will be within a few per cent of those of the network when the original distributions are used. In other words, the exponential assumption seems to be less important than the structure of the network: it is more important for the network structure to reflect that of the real sysem than it is for the distributions to reflect those of the real system. This property permits the use of exponential assumptions as a viable first approximation, thus yielding a network whose analysis is both tractable and useful. The results of analysis of queueing networks as described above have confirmed that in many cases (those listed above) the equilibrium state probabilities of the network model depend only on the means of the service time distributions and not on the exact distribution chosen.

The results outlined above have been concerned with determining equilibrium state probabilities. Research on other approaches to the analysis of queueing networks is currently underway. These other approaches include the application of the diffusion approximation to networks [4] and numerical solution techniques [5]. No single approach dominates the others. There are the usual tradeoffs as to the range of models to which they apply, the predicted performance measures, form of solution, etc.

The results for arbitrary networks have at this point a number of limitations (more optimistically we could refer to these as areas for future research). Some of these are:

1. Very little is known about the transient behavior of networks. The diffusion approximation approach has yielded some results on transient behavior but is thus far limited in the class of networks to which it applies.
2. Most studies have assumed a customer requests only one resource at a time.
3. Blocking effects in networks have proved difficult to handle analytically.
4. Studies of configurations subject to oscillatory behavior (e.g., thrashing) have just begun(8).

The analysis of multiple resource models is progressing rapidly (see [6] for an extensive survey). The results thus far have been promising but more experience is needed in the application of these models to real systems.

2. References

[1] Lassettre, E. R. and Scherr, A. L., "Modelling the Performance of the OS/360 Time-Sharing Option (TSO), in Statistical Computer Performance Evaluation," Ed. Walter Freiberger, Academic Press, 1972, pp. 57–72.

[2] Moore, C. G. III, "Network Models for Large-Scale Time-Sharing Systems," Technical Report No. 71–1, Department of Industrial Engineering, The University of Michigan, Ann Arbor, Michigan, April, 1971.

[3] Sekino, A., "Performance Evaluation of Multiprogrammed Time-Shared Computer Systems," MIT Project MAC Report MAC–TR–103, September, 1972.

[4] Koboyashi, H., "Application of the Diffusion Approximation to Queueing Networks: Part I—Equilibrium Queue Distributions," Proceedings of the ACM SIGME Symposium on Measurement and Evaluation, Palo Alto, February, 1973, pp. 54–62.

[5] Irani, K. B., and Wallace, V. L., "On Network Linguistics and the Conversational Design of Queueing Networks," JACM 18 (4), 1971, pp. 616–629.

[6] Baskett, F. and Muntz, R. R., "Networks of Queues, Seventh Annual Princeton Conference on Information Sciences and Systems," Princeton University, March, 1973.

[7] Buzen, J., "Queueing Network Models of Multiprogramming," Ph.D. Thesis, Division of Engineering and Applicd Science, Harvard University, Cambridge, Mass., 1971.

[8] Courtois, P. J., "On the Near-Complete-Decomposability of Networks of Queues and of Stochastic Models of Multiprogramming Computer Systems," Ph.D. Thesis. Computer Science Department, Carnegie-Mellon Univcrsity, Pittsburgh, Pa., 1971.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, Computer Performance Evaluation; Report of the 1973 NBS/ACM Workshop, March 27–30, 1973, San Diego, Calif. (Issued August 1975)

An Analytic Framework for Computer System Sizing and Tuning

Stephen R. Kimbleton

The RAND Corporation,* Santa Monica, California 90406

Performance analysis, as practiced by end users, appears to be dominated by the trial and error approach. Computer systems modeling techniques have received relatively little usage by such users except for the sporadic application of commercially available computer system simulators. Vendors, by contrast (cf. the various ACM and IEEE publications) have been extensive users of both analytical and simulation based techniques for performance analysis. This paper discusses some of the reasons underlying the lack of extensive usage of such techniques by end users, identifies an area of performance analysis appropriate to the usage of modeling techniques and discusses an approach to its investigation through their usage.

Key words: Analytical; computer systems modeling; end users; performance.

1. Utility of Modeling

End users are naturally wary of computer systems modeling techniques in view of the substantial initial investment required (development, calibration, verification, validation and training) and the absence of an existing structure which identifies the role of computer system modeling techniques in performance analysis. Since such a role has not been clearly defined, evaluation of the cost effectiveness of modeling is difficult and the trial and error approach is adopted by default.

Determination of computer systems modeling cost effectiveness requires careful identification of both costs and rewards. Some insight into the magnitude of these costs may be obtained by observing that although the yearly rental cost of the commercially available computer system simulators is in the neighborhood of $10,000, management usually "ballparks" the cost of a simulation based approach to performance analysis at $50,000.

Since the commercially available computer system simulators have been carefully engineered for ease of use by persons unskilled in modeling techniques, and since analytical techniques and other tools described in the literature have received relatively little human engineering, it follows that the cost of their application will be even greater. Thus, their usage has normally been restricted to universities and other technically oriented institutions in which the necessary skills are more readily available.

Two major rewards accruing through the use of modeling are: (1) greater insight into the nature of the basic physics of the process being modeled, and (2) reduced costs achieved through economies of scale permitted by widespread application of techniques developed at one location to many different locations.

In view of economic considerations, insight, although deemed a desirable byproduct, is usually judged significantly less important than reduced costs. However, in view of the incomplete state of many of the models described in the literature and the consequent need for additional development and refinement, cost reductions are difficult to demonstrate in the context of a single installation. Further, the opportunity to achieve economies of scale through simultaneous implementation of centrally developed models has been limited since each site within a given organization has usually been established as a unique entity.

Recently, a desire to achieve economies of scale in several areas of computing has led to unified procurements of relatively large numbers of homogeneous (i.e. same vendor, same product line) systems such

*Now located at Information Sciences Institute, Marina Del Ray, Calif. 90291.

as the Air Force Advanced Logistics System (ALS) and Base Logistics System (BLS) as well as the DOD World Wide Military Command and Control System (WWMCCS). Further, several of the larger corporations are either actively contemplating or are initiating such acquisitions. Such systems would appear to permit the necessary economies of scale required to underwrite the development and implementation of appropriate modeling techniques provided their role and a reasonable approach can be suitably identified.

2. Computer System Performance Components

Development of a role for modeling can be achieved through consideration of the major determinants of computer system performance which are: (1) the system workload, i.e. the collection of jobs to be processed, (2) operating personnel proficiency and (3) system configuration, i.e. the hardware and hard software which are to be used as determined by the sizing and tuning methodologies employed.

Identification of the proper system workload is a difficult task although substantial economies can be achieved by elimination of redundant or unnecessary jobs [BELLT 72][1]. Centralization of the investigation of proper system workload appears unlikely since it requires careful systems analysis at each site.

Improvement of personnel proficiency is an organizational problem. As such, it requires careful attention to prevailing attitudes of the personnel involved, incentives available to improve performance and management willingness to adopt or implement more flexible approaches to achieve improved performance. Because of the complexity of this problem, most installations are more concerned with achieving a satisfactory level of personnel proficiency than with optimizing personnel proficiency.

Optimization of the system configuration as a function of the workload and assuming acceptable personnel proficiency requires consideration of the performance impact of: (P1) file locations, (P2) hardware configuration, (P3) job resource requirements and reference behavior, and (P4) the schedule detailing the sequence in which the execution of jobs is to be initiated. These topics are appropriate for investigation through the use of computer systems modeling techniques. However, the difficulty of achieving useful results from a management viewpoint is increased by the (performance) interrelationships existing among these variables. Thus individual studies of a single variable provide information of uncertain value; studies providing information on the interrelationships among these variables would appear to be more useful.

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

3. Using Modeling in Sizing and Tuning

We now discuss two endpoints of a spectrum of possible model based approaches to obtaining 'good' system performance as a function of P1-P4. These approaches would be implemented through either: (1) application of currently existing optimization techniques available within the operations research literature, or (2) development of a heuristic procedure for performance improvement based upon computer systems modeling techniques.

The first approach, although initially attractive, suffers from the severe defect that the extensive simplifying assumptions required for the application of optimization techniques, e.g. mathematical programming [DANTG 63], markov renewal programming [HOWAR 60] or other optimization techniques [WILDD 67] renders interpretation of the results obtained difficult. Typically, at least one of the following simplifying assumptions is made: one job at a time in execution (contrary to the multiprogramming nature of current computers), job processing times are deterministic (contrary to the observed interdependencies between job processing times and the collection of concurrently executing jobs) or conceptualization of the system as a single resource (contrary to the multitude of interactions among device categories, e.g. processors, executable memory, secondary storage devices and data paths).

The unsatisfactory nature of the results achieved through this approach is implicit in the failure of management to make significant use of either the extensive existing literature on scheduling in general (cf. [CONWR 67]) or its application to computer system scheduling in particular (cf. [SAHNV 72]). To achieve results of more direct use to management, the modeling techniques employed should permit a closer rendering by the model of the dynamic interaction among jobs, systems and schedules. This can be achieved provided one is willing to forego direct application of existing optimization techniques and is, instead, willing to consider development of heuristic

approaches. The basic components of such an approach are (1) a means of determining system performance given P1-P4 have been specified, and (2) techniques for determining desirable modifications to achieve systems with better performance.

Evaluation of changes in system performance through direct modification is usually unsatisfactory because of the time required for implementation. Studying the performance effects of changes through simulation models can usually be performed more rapidly. However, although simulation is faster than direct modification, a computer system simulator which provides reasonably extensive information on device utilizations and delays (required for determining the nature of subsequent modifications) typically operates at a rate of 2–10 times faster than real time. This is clearly too slow to permit testing of large numbers of modifications which would require speeds of at least two orders of magnitude faster than real time.

Several analytic approaches to the prediction of computer system performance have been developed [GAVED 67], [MOORC 71], [BUZEJ 71,] [KIMBS 72]. Although these approaches appear capable of achieving the required speed, they typically assume that the workload can be probabilistically specified and that jobs are either statistically homogeneous or can be classified as being from one of a collection of statistically homogeneous job classes. Provided this restriction can be suitably relaxed, one approach to achieving the required speed would be through the use of analytic models to predict system performance over those periods of time during which the composition of the mix is constant and then to aggregate the resulting performance information to determine shift performance.

To evaluate the utility of this approach, a prototype version of such a performance prediction tool has been implemented [KIMBS 73]. The initial prototype could process a shift of N jobs in 3N seconds (on an IBM 360/67). This would seem to be a reasonable speed for production batch installations. Examination of a few test cases revealed that agreement between predicted and measured values was reasonable, e.g. the difference was in the range of 15–20% of the measured value. Through a process of refinement of both the analytic techniques used and optimization of the code produced, an additional increase in speed should be achievable. Thus, a potential means for satisfying the first requirement for the development of a heuristic approach to the performance analysis of production batch systems is at hand.

The second requirement for development of a heuristic approach for achieving "good" computer system performance is a means for determining the nature of desirable modifications. This requires identification of a quantitative interpretation of the term "desirable," and, as will be attested to by anyone who has ever engaged in a performance analysis effort, is a difficult problem whose general solution may exist only in parametric form. However, given that a quantitative interpretation has been chosen, the currently existing computer systems modeling literature would appear to provide extensive information for the development of suitable heuristics.

4. Concluding Remarks

It is clear that development and implementation of an approach such as that described would tax the resources of an individual site. However, once such an approach has been initially implemented, its transfer to and usage by individual sites should be cost effective.

Modeling is also useful because it forces consideration in a quantitative manner of the basic forces driving the process being modeled. As an example, determination of suitable workload characterizations and desirable monitor requirements are topics of continuing concern. Answering these questions in this generality would appear to be exceedingly difficult, if not impossible since no means has been provided to test the adequacy of the answer. However, if instead one requires only that the workload characterization, used in conjunction with a suitable modeling approach should yield some specified level of agreement between predicted and observed values, the question assumes a more tractable form. The potential existence of an answer is suggested by the capability of commercial simulators to yield reasonable results in many cases. It should be noted that the determination of monitor requirements can then be regarded as an exercise in evaluating their capabilities for gathering the data required for the workload characterization. Although this approach does oversimplify matters somewhat, it does indicate that answers to these questions exist provided the questions are posed in more specific, and thus answerable terms.

Preparation of this paper was supported in part by the Office of Naval Research under contract N00014–67–A–0181–0036 (NR 049–311). The author would like to thank Messrs. K. S. Das, S.-C. Lin, V. Jeyabalan and Ms. C. Stallings who participated in the preparation of this paper.

5. References

[BELLT 72] Bell, T. W., B. W. Boehm and R. A. Watson, "Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort," The RAND Corporation, R–549–1–PR, November 1972.

[BUZEJ 71] Buzen, J. P., "Queueing Network Models of Multiprogramming," Ph.D. Diss., Harvard University, August 1971.

[CONWR 67] Conway, R. W., W. L. Maxwell and L. W. Miller, "Theory of Scheduling," Addison-Wesley, Reading, 1967.

[DANTG 63] Dantzig, G. B., "Linear Programming and Extensions," Princeton University Press, 1963.

[GAVED 67] Gaver, D. P., "Probability Models for Multiprogramming Computer Systems," *J. of Assoc. for Comp. Mach.*, Vol. 14, No. 3, July 1967.

[HOWAR 60] Howard, R. H., "Dynamic Programming and Markov Processes," MIT Press, 1960.

[KIMBS 72] Kimbleton, S. R., "Performance Evaluation—A Structured Approach," AFIPS Spring Joint Computer Conference, Vol. 40, May, 1972, pp. 411–416.

[KIMBS 73] "Analytic Approaches to Fast Computer System Performance Prediction," in preparation.

[MOORC 71] Moore, C. G., "Network Models for Large Scale Time Sharing Systems," Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan, April 1971.

[SAHNV 72] Sahney, V. K., and J. L. May, "Scheduling Computer Operations," Computer and Information Systems Division, American Institute of Industrial Engineers, Norcross, 1972.

[WILDD 67] Wilde, D. J., and C. S. Beightler, "Foundations of Optimization," Prentice-Hall, 1967.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION;
REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Relating Time and Probability in Computer Graphs

Robert R. Johnson

Burroughs Corporation, Detroit, Michigan 48232

Program behavior is discussed in terms of Petri Nets. Using probabilistic information, an experimental method is given to determine the time spent in each state. To illustrate the method, two examples are given for program graphs.

Key words: Evaluation, graphs, measurement, networks, performance, Petri Nets, pobability.

Computer behavior can be represented by directed graphs in general and by Petri Nets in particular with each state labeled with its probability of occurrence.[1] For any given net, these state probabilities can be computed:

a. Under Bayesian (or equilibrium) conditions (where the conditional probabilities for each of the n states following immediately each event are equal to $\frac{1}{n}$)

or

b. Where some or all of the conditional probabilities are known from other experimental or prior information, and the unknown conditional probabilities are treated under Bayesian conditions.

The equilibrium state probabilities for each state on that Petri Net which represents the whole (problem + machine) structure can be computed noting that the sum of the state probabilities for each participant must equal 1. This is to say that each program participant is always at some one place in the net. For only one program, this simply states that the program continually "restarts" whenever it "completes" so that it is always running, progressing through the net or exiting the net and reentering it.

[1] Johnson, Robert R., "Some Steps Toward an Information System Performance Theory", first USA-Japan Computer Conference, Tokyo, Japan, Oct. 1972.

The expected or "typical" state probabilities for any given set of data exercising any given Petri Net can be experimentally determined.

The object of this note is to present an experimental means to determine the time spent in each state. With this information and information about the probabilistic behavior of a new data base, the time to execute this new data base can be predicted. The concepts involved here are defined with respect to the events and states of a Petri Net:

$n_i \equiv$ number of occurrences of the state i. State i follows event i

$p_i \equiv$ probability of the state i

$$p_i = \frac{n_i}{\sum_i n_i} \text{ where } \sum_i p_i = 1 \qquad (1)$$

$t_i \equiv$ per unit time spent in state i per occurrence. This assumes time spent in state i is the same for each occurrence of state i.

$U.T. \equiv$ unit time for the equilibrium occurrence conditions n_{ie}:

$$U.T. = \sum_i n_{ie}\, t_i \qquad (2)$$

$A_i \equiv$ appearance of occurrence of state i measured as the proportionate time spent in state i out of total possible time:

$$A_i = \frac{n_i\ t_i}{\sum\limits_i n_i\ t_i} \tag{3}$$

$T_j \equiv$ execution time required to traverse the series of states i for an experiment j:

$$T_j = \sum_i n_{ij}\ t_{ij} \tag{4}$$

Time can be related to the state probabilities by approximating the state probabilities with their appearance:

$$P_i \cong A_i \tag{5}$$

The appearance of a state i can be measured experimentally:

$$A_i = \frac{n_i\ t_i}{\sum\limits_i n_i\ t_i} \tag{6}$$

For an experiment j, the appearance A_{ij} is computed:

$$A_{ij} = \frac{n_{ij}\ t_{ij}}{\sum\limits_i n_{ij}\ t_{ij}} \tag{7}$$

For stationary processes, where outside real time events do not change the times required to execute the events, the t_i can be assumed independent of the experiment $_j$:

$$t_i = t_{ij} \text{ for all } j \text{ considered here} \tag{8}$$

The per unit state times t_i can be determined by running a number of experiments j for different induced values n_{ij} and measuring total elapsed time T_j:

$$T_j = \sum_i n_{ij}\ t_i$$

$$t_i = \left[\sum_i n_{ij}\right]^{-1} T_j \tag{9}$$

This requires $j = i$ number of experiments and the knowledge of the n_{ij} for each experiment.

The number of occurrences n_{ij} is computed by knowing (or setting) the conditional probabilities at each event (or branch point) in the graph, and making use of the relationship between probability $\underline{p}$ and conditional probability $\bar{p}$:

$$\underline{p}_i = \sum_k \underline{p}_i\text{-l},\ k \cdot \bar{p}_i \tag{10}$$

where there are k states preceding state i.

Two examples are given for the typical program graph shown in Figure 1. The first example is computed for equilibrium (or Baysian) conditions when the conditional probability for taking each branch is $\frac{1}{2}$. The second example is computed for the case where the conditional probability for taking the loop branch is 50 times the conditional probability of taking the exit.

The probability of the top state in each example is chosen as X and each subsequent state probability is determined in terms of X. Unknowns Y and Z are introduced for the two top loops and solved:

For state 02:

$$p_2 = Y + Z$$

For state 03:

$$p_3 = \frac{1}{2}\left(Y + X + Z\right)$$

but also:

$$Z = \frac{1}{2}\left(Y + X + Z\right)$$

$$\therefore Z = X + Y \tag{11}$$

For state 04:

$$Y = \frac{1}{4}\left(X + Y + Z\right)$$

and using (11)

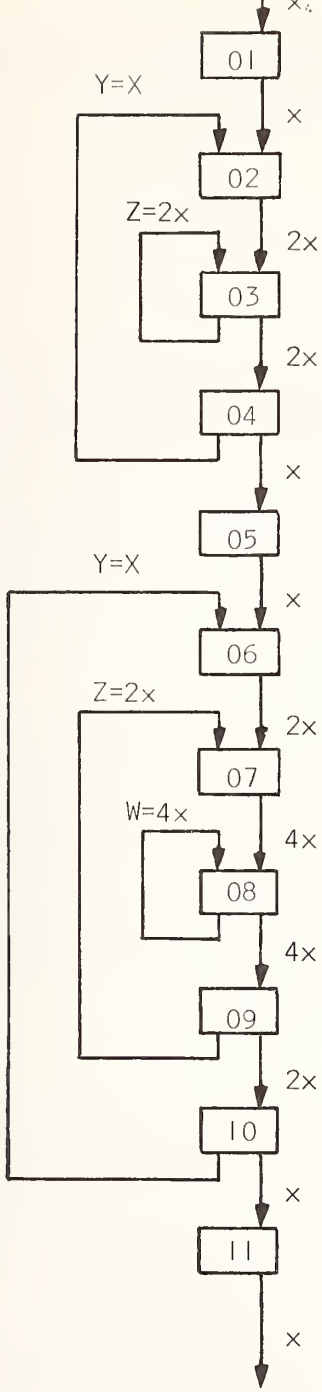
$$Y = X \tag{12}$$

Thus:

$$Z = 2X$$

$$Y = X$$

as shown on the graph. This same method is used to compute probabilities as shown on the rest of each graph.

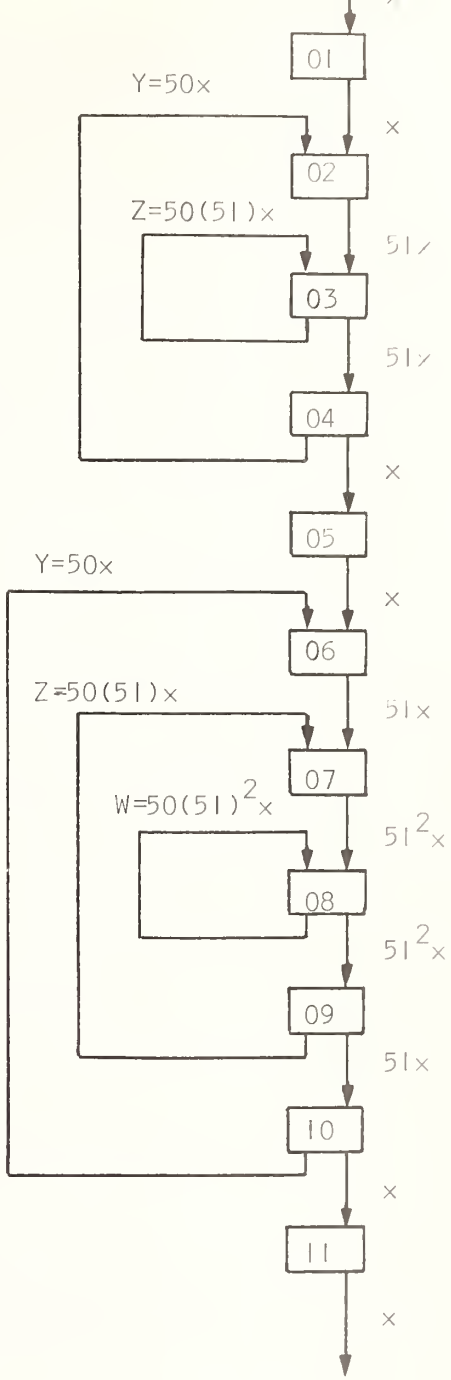


Equilibrium Conditions

Each loop taken once per exit

$$X = \frac{1}{31}$$

$i = 16$



Known Statistics

Each loop taken 50 times per exit

$$X = \frac{1}{140661}$$

$i = 16$

Figure 1

```
          B6700/B7700   FORTRAN   COMPILATION   MARK   2,3,017
$$LIST SINGLE STACK CODE FREE OPT = 1
$SET  LONG
$SET  VECTORMODE
$SET  GRAPH
        DIMENSION A(50.50), B(50.50), C(50.50)
        N=50
        DO 10 I=1,N
        DO 10 J=1,N
        A(I,J)=SIN(,4)
        B(I,J)=SORT(2.0)
10      CONTINUE
C*
C*        THE ABOVE LOOPS ARE TO INITIALIZE THE ARRAYS 'A' AND 'B' TO
C*        NON-ZERO VALUES, TO KEEP THE ADDS AND MULTIPLYS FROM BECOMING
C*        TRIVAL.
C*
        WRITE(6,65)
1000    CONTINUE
        TI=TIME(11)
        DO 200 II=1,N
        DO 200 JJ=1,N
        C(II,JJ)=0
        DO 100 K=1,N
        C(II,JJ)=C(II,JJ)+A(II,R)+B(K,JJ)
100     CONTINUE
200     CONTINUE
        T1=(TIME(11)-T1)+2.4/1000,0
        WRITE(6,66) N,N,T1
        N-N-10
        IF (N,I,1.0) STOP
        N=MAXO(N,1)
        GO TO 1000
65      FORMAT(///)
66      FORMAT( ' TIME FOR ', 12, ' BY ', T2,
        END     ' MATRIX MULTIPLY IS ',F8,2, ' MILLISECS.')
```

FIGURE 2.

The value of X is determined by summing the state probabilities, equating to unity, and solving for X as shown.

The n_i for state 08 of the first experiment (equilibrium) is 4 and $\Sigma\, n_i$ for this experiment is 31. For the second experiment, n_i for state 08 is 51^2 and $\Sigma\, n_i = \frac{1}{140661}$.

These values of n_i and $\Sigma\, n_i$ are computed for $i = 16$ different experiments, the program is executed those 16 different times and its execution speed T_i measured. Using these values, the per unit state times t_i are computed by equation (9). The program shown in Figure 2 has the graph of Figure 1.

Summary

A practical experimental means to compute the per unit time spent per state in a program is presented. This means requires measuring elapsed time of the entire program only, plus being able to preset the number of times each branch is taken during experimental runs.

These per unit times then are used (via equations 1, 4, 5) to predict the program's execution time when only probabilistic information is known about the program's data base.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION; REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

On the Power and Efficiency of a Computer

L. Hellerman

International Business Machines Corporation, Poughkeepsie, New York 12602

The concept of power is defined and proposed as a new performance measurement tool for computer systems. Several examples are given that illustrate the calculation of power for small devices. The efficiency of a system is then discussed in terms of power. Finally, the new methods are compared with other methods of system evaluation.

Key words: Computer efficiency, evaluation, measurement, performance, power, work.

1. Introduction

Elsewhere [1][1] we outlined a theory of computational work. The computational processes we dealt with were thought of as being implemented by an organization of steps, and the work of each step was determined from its truth table, or more generally from its table-lookup specification. The whole theory was based on the definition of the work of a step $f{:}X \to Y$,

$$w(f) = \frac{|X|}{|X_i|} \log \frac{|X_i|}{|X|}$$

where $Y = \{y_1, \ldots y_n\}$, $X_i = f^{-1}(y_i)$, and $|X_i|$ is the number points in the set X_i. We investigated some properties of this measure, such as the dependence of work on the kind of implementation (Cartesian, compositional, and sequential). We showed that the measure could be interpreted as the information in a memory for the table-lookup implementation of the step, so the unit of work is the unit of information. To distinguish the context, we could not resist the suggestion that the work bit be called a "wit," and the unit of power—a wit per second—be called a "wat." Except for this we said nothing at all about the time for execution of a process.

2. Computational Power

Real processes implemented on devices or facilities take time. A device or facility that does computational work w in time t, the cycle time, may be said to have a power rating of w/t. This seems simple, but there may be a problem when we try to reconcile the overall power of a collection of devices in a facility with the sum of the powers of the individual devices. Suppose a process consists of two individual sequential steps: Step 1 doing work w_1 in time t_1, and Step 2 doing work w_2 in time t_2. Looking at the process as a whole, the total power is $(w_1 + w_2)/(t_1 + t_2)$. But looking at the individual steps, the power is $w_1/t_1 + w_2/t_2$. These are not the same.

Which shall we take as the power of the process? The answer is clear as soon as we see that the device for Step 1 is utilized only during the time interval t_1. Although its rating over this interval is w_1/t_1, its rating over the process cycle time is $w_1/(t_1 + t_2)$. Similar considerations for Step 2 then show that the overall rating of the facility is $(w_1 + w_2)/(t_1 + t_2)$. Thus, we have a Principle: The power of a facility comprising a set of devices is the total work per total time, provided each device is used just once in each use of the facility. This principle is independent of and particular work measure $w(f)$. We find it convenient in applications of our measure, but it would be just as convenient using any other measure.

The provision that each device be used just once is clearly met by the Cartesian and compositional synergisms. To be met by the sequential synergism, in which the first step is a branching process g which selects

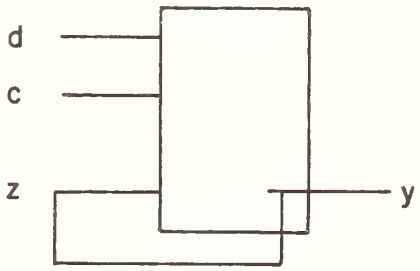
[1] Figures in brackets indicate the literature references at the end of this paper.

just one of a set of possible alternative second steps $\{f_i\}$, it is necessary to look upon the set $\{f_i\}$ as a single device. We shall take this view.

We shall now apply the principle in evaluating the power of some facilities.

3. Application: Gated Latch

Consider the gated latch, with truth table shown.



d Data
c Control
z Internal State
y Output

c	z	d	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

The device may be implemented sequentially as follows.

Step	Work
Branch on c	2
c=0 : y ← z	2
c=1 : y ← d	2

In this case, the sum of the works of the steps is 6. By the principle, if the cycle time is t, the power is $6/t$.

4. Application: Decoder

Suppose a decoder of n binary variables $x_1\ x_2, \ldots, x_n$ operates sequentially. The first step branches on x_1

$$f(x_1) = \begin{cases} f_0 & \text{if } x_1 = 0 \\ f_1 & \text{if } x_1 = 1 \end{cases}$$

then:

$$f_0(x_2) = \begin{cases} f_{00} & \text{if } x_2 = 0 \\ f_{01} & \text{if } x_2 = 1 \end{cases}$$

and:

$$f_1(x_2) = \begin{cases} f_{10} & \text{if } x_2 = 0 \\ f_{11} & \text{if } x_2 = 1 \end{cases}$$

and so on. Since each operation takes work 2 and since there are 2^n-1 operations, the work of the decoder is $2(2^n-1)$. There are n levels to the decoder, and if the entire process takes time t, each level takes time t/n. But each level has utilization $1/n$, so the power of the decoder is $2(2^n-1)/t$, as expected from the Principle.

5. Application: Read Write Storage

A storage with 2^m words (m address bits) and n bits per word may be thought of as comprising an m bit address decoder, and $2^m n$ storage cells, organized as 2^m words, each with n bits. The j-th bit of the i-th word may be implemented by a gated latch as shown below: Here d_j is the output of the j-th bit of the storage data register (SDR) and y_{ij} is an input to the j-th bit of this SDR. In [1], we saw that the work of the two variable AND is 3.245. Hence, the work of the storage is:

$$w = 2(2^m-1) + 12.49\, n2^m$$

A few values are given in Table 1. The IBM System/360 G40 storage, having 16 address bits, 16 content bits per word, and an access time of 2.5×10^{-6} seconds, has a power of 5.29×10^{12} wats.

TABLE 1.—*Work of a storage, in wits, 2^m words, n bits per word*

	m		
	16	24	32
n	x10^6	x10^9	x10^{12}
16	13.2	3.39	0.867
32	26.3	6.74	1.73
48	39.4	10.1	2.58
64	52.5	13.4	3.44

6. The Power and Efficiency of A System

We think of a computer system as comprising a set of facilities—$F_1, \ldots, F_n$. Each facility may be a storage, a register, an incrementer, a mover, an arithmetic-logic unit, ... anything that has input states and consequent output states. We have indicated by a few examples how the work and power of a facility may be determined. The aggregate power of a system is defined as the sum of these:

$$A = \sum_{i=1}^{n} \frac{w_i}{t_i}$$

When facilities are interconnected in a particular system, they do not work continuously, since each must wait for information from the others before it has something to do. Because of this, given an interconnection and a workload there is a u_i for each facility, $0 \leq u_i \leq 1$, giving the utilization of the facility. The utilization may be determined from an analysis of the system as a queue network, or from the control program, or any other means. The power of the system on the given load L is then:

$$P(L) = \Sigma \frac{u_i\ w_i}{t_i}$$

and the efficiency of the system under this load is:

$$E = \frac{P(L)}{A}$$

7. Summary

We wished to show that the computational measure proposed in [1] can be used to evaluate the power and efficiency of real systems. To be most convincing, we should take a real system and evaluate it, (as in [5]), but this is out of the question with our limited pages. Instead, we took a representative facility (a storage) and showed how to evaluate its work and power, and how to use these results in the overall evaluation of a system.

8. Discussion

System evaluation in terms of "wats" does not have the mix weakness of mips; nor is it magically derived, like some other measures. Still, it may be asked: What is its use? Why do we need such a measure? We have come to look upon the problem of measuring processes as a key technical problem of the computer industry. It has plagued us for many years. In 1960, Bagley [2] wrote: "There is a need for a measure of data processing ability * * * " Seven years later, Calingaert [3] wrote: "The accurate estimation of the performance of a specific system on a given application is clearly a very difficult problem * * * It is a challenge which must be met, however, if the computer industry is to achieve maturity." And in 1969, Emanuel Piore [4] was reported to have urged the academic community "to articulate these industry requirements for a theoretical base, for a set of measurements for hardware and software." Still, industry seems to have gotten along in the past, somehow, without such a theoretically based measure. Is it really needed? What would we do with it? Replace mips? Is there something in the course of the industry's development that makes the problem more urgent at present?

Perhaps. We may be guided by analogy with physical energy. The trend toward systems in which many users share resources, toward networks of computer installations, * * * obviously resembles power distribution systems. Just as an electric power installation generates and distributes power to customers who run machine tools and washing machines, toasters and lights, similarly computer installations generate and distribute computational power to compute payrolls and insurance premiums, make reservations and execute APL transactions. Just as the motors, toasters, and lights have their power ratings, and draw power

from the generator, similarly, the payroll, reservation, and APL transactions draw computer power from the system. How much? The question is important because, presumably, the cost to the user should be proportional to the amount of power he uses, his "watminutes". This is very different from the time the user is on the system.

Consider: If two jobs are in a multiprocessing system for the same time, but one job uses a weak combination of resources while the other uses a powerful combination, should not the cost of processing the jobs reflect this?

Measurement is needed when there is an exchange, a buying and selling, a taking and transferring of computational power. And now, we see it is not enough for a power measure that it be applicable only to the internals of a computer system. The measure must also be applicable to the jobs that are processed on the system. The statements of user oriented languages such as FORTRAN, COBOL, and APL, all specify processes in their own right, conceptually, existing independently of their hardware implementation in some system. Can we measure the work of these processes in the same terms used to measure the computer? Yes. It does not matter whether the inputs and outputs of a process are distinct hardware states, or distinct conceptual entities; the principle of the measure is the same. So the answer is yes, in principle. But the systematic application of the measure to all possible statements of a language like APL in order to measure jobs expressed in that language is another matter. This is what remains to be done.

9. References

[1] L. Hellerman, "A Measure of Computer Work," IEEE Trans. on Computers, Volume C–21, No. 5, May, 1972, pp. 439–446.

[2] P. R. Bagley, "Two Think Pieces," Commun. Asso. Comput. Mach., January, 1960, Volume 3, No. 1, p. 1.

[3] P. Calingaert, "System Performance Evaluation: Survey and Appraisal," Commun. Asso. Comput. Mach., Volume 10, January, 1967, pp. 12–18.

[4] E. R. Piore, quoted in Datamation, October, 1969, p. 161.

[5] L. Hellerman, The Power and Efficiency of a Computer System, (to appear in the) Proceedings of NTG/GI—Fachtagung Struktur und Betrieb von Rechensystemen vom 20.—22. März 1974 in Braunschweig. This paper gives an evaluation of the IBM System/360 Model 40 in terms of the measure in [1]. The main results are:

Aggregate power of the system is 2.565×10^{12} wats.

Efficiency of instructions ranges from .63 to .98.

Work of instructions ranges from 142×10^3 to 1068×10^3 wits.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406 COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

The Weltansicht of Software Physics

Kenneth W. Kolence

Palo Alto, California 94303

This paper is a brief exposition of the idea that a "software physics" exists, and furthermore that it is based on the same concepts as used in the natural sciences. The idea of a software unit is introduced to name the entities embodying the basic observable properties of software physics. These properties are identified as work and time. (Another property, existence, is not referenced in this paper.) The relation of these properties, in a general sense, to the variables of performance monitors and modeling is commented on.

Key words: Computer performance measurement, software physics, software units, software work.

To a large extent, the way one looks upon the basic nature of an object of scientific study determines the form and the power of the subsequent theoretic constructs. Examples of this fact abound in the history of science. It therefore is important to be explicit about one's world view when proposing the development of a field of science. The purpose of this paper is to explicitly consider the world view from which I believe a software physics can be fruitfully constructed.

It is impossible for me to start with the "most important" idea first, since this is one of those situations where several ideas or concepts appear to be of equal importance. In a way, one may also look upon the set of concepts as a set of axioms, some of which may be related or replaced with others to build different logical constructs. But the one concept which appears to justify the use of the terminology "software physics" and which certainly strongly affects the form of the theory arises from the following observation.

There are many fields of human endeavor which call themselves sciences: physics, chemistry, biology, sociology, anthropology, and computer science are but a few. Of these, we observe that those which use the basic "principles" of physics and chemistry make up a conceptually single group, characterized by common terminology and, further, the ability to translate knowledge in one special area to other specialized fields. We tend to call these the "hard sciences" or the "physical sciences." The other group is most singularly characterized by their individual iconoclasm.

Computer science is absolutely unrelated to economic theory, sociology, political science, etc. Indeed, even in that subgrouping of the "soft sciences" which is generally concerned with the study of man and his behavior, little if any interconnection exists in the deep conceptual sense found in the physical sciences. As a means of differentiating between these groupings of sciences, let us divide sciences into "physical sciences" and "singular sciences," where the term singular is used to denote the lack of an underlying conceptual infrastructure between the singular sciences, in opposition to the infrastructure of the physical sciences. (The infrastructure of the physical sciences is well illustrated in Margenau's "The Nature of Physical Reality," McGraw Hill.)

A fundamental choice in one's world-view consists in the (usually implicit) decision to either approach the building of the science from a singular point of view, or within the context of the infrastructure of the physical sciences. An absolutely essential point to understand is that one must make this decision wholeheartedly: either the complete infrastructure is accepted or not. To "borrow" terms and ideas from the physical sciences without acknowledging the full conceptual linkage between all of the fundamental concepts of the physical sciences is to straddle the fence between the physical sciences and a singular science. As the 18th and 19th century Rationalists discovered, the transition probability on that fence is heavily biased toward the singular sciences.

The fundamental choice of software physics is to wholeheartedly accept the full conceptual infrastructure of the physical sciences as the foundation from which to evolve a theory of softwear behavior.

There are good, practical reasons for this choice. If the choice is wrong, we shall be forced into the field of singular sciences relatively rapidly—say, less than a decade. But, we shall know why we are a singular science, and have at least some proof, namely our failure, that we are indeed singular. But, if the choice is right, we will be deeply aided and speeded on our way by the availability of the infrastructure and its intellectual wealth of preciseness and form. Analogy, that most powerful of tools of scientific discovery, is at the same time a most dangerous of seas to venture upon for a scientific quest. It is less dangerous if the full conceptual infrastructure is accepted as one can subject the analogy to at least some critical tests. For singular sciences, the shoals of analogy are uncharted.

Another practical reason for our choice is that our decision permits us to recognize theory when we see it, as opposed to accepting mechanistic descriptions as theory—the bane of the soft sciences. We shall return to this point in more detail later.

If this fundamental choice names software physics, and clearly and cleanly separates it from the field of computer sciences as it is known today, it does so intellectually but, of course, not in terms of the object of study. Yet, the world-view of software physics has yet another important concept which both broadens and simplifies the object of study beyond that of computer sciences: the software unit.

In software physics, the object of our study must be the inherent properties of software, without regard to the arbitrarily selected sizes or packages of code which we name subroutines, tasks, programs, jobs, applications, operating systems, etc. In other words, we must be at least initially concerned with universal properties of software. Each of the aforementioned packages may well have interesting properties in their own right, but the properties of first interest to software physics are those which they all share. A word, a name, is needed to characterize this set of universal properties, and any grouping of code which may be of interest in the context of such properties. The name I have selected is software unit. Thus, whenever in software physics one speaks of a software unit, one is not distinguishing size; rather, one is distinguishing universal properties from properties arising uniquely from the structural, and perhaps functional, choices made during the design process.

The softwear unit plays a role in software physics roughly the same as the center of mass, in its role as a point mass, plays in natural physics. In fact, throughout the natural physics, one deals with equivalent concepts; electrical charge, time, mass, energy, forces, etc. are universal properties associated with matter in some sense, regardless if matter is artificially fashioned into an object or if it is considered in terms of molecules or galaxies. The term software unit is meant to convey the vessel in which similarly universal properties are embodied.

One great advantage of the software unit concept is that the properties of software units are observable to monitors and other forms of instrumentation. In fact, with extremely few exceptions, the observables of computer monitors are only observables of software units. The current basic challenge of software physics is to provide a basic unifying theory relating these observables one to another in meaningful ways. This work has been completed in essence, and is currently being prepared for publication. The work to be published must be experimentally tested before it can be called an accepted theory. At the minimum however, it will represent an example of a theory in software physics. Thus, the essential aspects of the software physics world-view can be summarized by saying that it is believed the basic principles and concepts of the natural sciences will be found to apply to the behavior of the universal properties of software units. It should perhaps be explicitly pointed out that descriptions of software, such as listings, flow-charts, etc., are outside of the current range of interest of software physics.

Software units assume the physical form of electrical and magnetic states within a computing system, and the observables of software physics, such as "CPU busy," are due to the action of software units within the computing system. In simpler words, one measures the effect of a software unit driving a computing system. An interaction thus exists between the workload software unit and the physical configuration of the computing system. This is most obvious when one considers a family of machines, such as the 360 and 370 series. Within a given machine type, say a 360/65, the I/O configuration attached may vary considerably. If one runs an identical program software unit on two or more 360/65's with different I/O configurations, one is apt to observe quite different

I/O measures. Yet the CPU measures as provided by, say PPE, are quite constant. Reversing the conditions and changing main frames up and down the 360 line, one obtains a variation in the CPU measures as well as I/O.

It is perhaps a subtle but important point that the same workload produces different values for the observables. It leads to the question of which observables are independent of configuration and which are at least partially, if not wholly, dependent upon the configuration in which the software unit is physically realized. In my work, the question has an especially simple answer: for a given software unit realized identically on two or more different configurations, the work done by a software unit is independent of configuration, but the times associated with performing that work are dependent on the configuration. Variables, such as power, composed of work and time variables, are dependent on the configuration through time, and independent with respect to work. This, by the way, is an easily testable hypothesis given precise definitions of the terms work and time.

Both work and time are also concepts of the natural physics. In software physics, work and time must be fully equivalent at the conceptual level to these concepts in natural physics if our world-view is to hold. As it turns out, in my studies at least, time has been the more difficult to be precise about. Work however is the key conceptual link between the natural physics and software physics, since it links directly to the concepts of energy and force, and thence on to the remainder of the conceptual infrastructure of the natural sciences. Regardless of the correctness of my own work, I would expect that the concept of work is the key to a demonstratively viable software physics.

Work, in my studies, is said to be done by a software unit whenever a medium is recorded upon, and the amount of work performed is numerically equal to the number of bits acted on. (This means the identity transformation does the same amount of work as a transformation which changes all bits.) In natural physics, work is performed whenever a force acts to change the state of the system under observation. These two definitions are equivalent, with the software unit playing the role of the force, and the media acted upon (e.g., core, registers, magnetic tape or disk, punched cards or paper tape, printer paper, etc.) representing the system under observation. Because of this equivalence, the definition of software work results in the identification of a software unit as a force because of the relationship between these two concepts in the natural sciences. The equivalence also forms a solid link, in my studies at least, between the two physics which will maintain the essential world-view belief that software physics is not a singular science.

Certain implications of the idea of a software physics are meaningful to the practical problems of computer measurement, and others to equally practical problems in the current efforts to analytically model computer systems behavior. The first set of problems are directly addressed by the work currently under preparation. Suffice it to say that most of the questions concerning the meaning and relationships between observables obtained by monitoring are resolved in very simple ways. However, the implications in terms of analytic modeling are not covered in that work, and a few words on the subject are useful here.

Perhaps the most fundamental implication, and one which nicely spotlights the distinction between modeling and theory development, lies in the choice one has as to the variables used in an analytic model. Currently, one normally assumes rather limited "workload distributions," and is completely free to select whatever variables appear appropriate. Because the analytic results one obtains often differ depending on the workload distributions used, and because these distributions are not known to generally occur in practice, the results of most modeling efforts are rather limited in their generality. More to the point, however, the variables selected (e.g., "mean arrival rate," "mean service time," "page fault rate," etc.) are unrelated to fundamental properties of software in general or, in our terms, to software units. Since they are fundamentally "time" variables, they are deeply related to a particular hardware configuration. By itself, this is not bad. What is bad is that they are not related often in a sufficiently analytic fashion to be generally meaningful.

In essence, current modeling efforts are hampered in attaining generality by two factors: no accepted theory exists which identifies the fundamental variables of software behavior, and no general method of characterizing workloads in terms of these variables is available. A theory, or more correctly, a sufficiently powerful theory of software physics should resolve these difficulties. My own work is but a step toward that sufficiently powerful theory, but hopefully it will be of some use in model building by both its world-view and its definitions of software unit work and time.

CHAPTER IV

Recommendations

A. INTRODUCTION

One of the objectives for the Workshop was to produce a set of recommendations on specific topics. Initial position papers on each topic were written by participants prior to the beginning of the Workshop. In addition to the four planned topics (Standards, Professional Activities, Research and Development, and Education and Training) two additional topics generated so much discussion during earlier sessions that they were separately discussed. These two topics were Monitor Register Standardization and Workload Characterization.

In order to address all the topics the participants met in separate panels and generated sets of recommendations based on the initial papers and the earlier sessions. Subsequently, all the Workshop participants discussed these recommendations and attempted to reach a consensus position.

The material in each of the following sections begins with a summary of the outcome of the discussions; these are repeated from the first chapter for the sake of clarity. The initial position paper on each topic follows for the topics in which such papers were written. Finally, excerpts from the discussions are provided in order to indicate how the participants arrived at the final positions.

B. STANDARDS

1. Workshop Results

Workshop participants were unanimously in favor of the following recommendations:

A representative organization such as the National Bureau of Standards (NBS), American National Standards Institute (ANSI), or Computer and Business Equipment Manufacturers Association (CBEMA) must formulate guidelines for:

1. Terminology;
2. A Minimum Set of Accounting Data;
3. Intrinsic Monitoring Capabilities for Computing Systems.

The word "guidelines" is a much weaker term than "standards." It implies a set of broadly-disseminated reference definitions which the community recognizes as nominal and preferred usage, but which are not universally binding. In some areas, such as the Monitor-Register discussed below, attempts were made at the Workshop to press toward advocating standards, but unanimity could not be achieved. In general, the reluctance to advocate standards was based on a feeling that the CPE field was not sufficiently well-understood yet. Standards developed at this time might not be sufficiently easy to apply, might be discriminatory, and might stifle innovation. Thus they might not be a net benefit for the field. In some areas, such as benchmark standardization, even guidelines were felt to be too ambitious.

However, there was a very strong feeling at the Workshop that significant performance losses result from the current confusion with respect to definition of such terms as CPU utilization, response time and overhead, with respect to proliferation of incompatible sets of accounting data, and with respect to definition of interfaces to monitoring tools. The recommended guidelines are needed as soon as possible. At the very least, their formulation will serve as a stimulus to improved communication in the CPE field, and in some cases they might serve as successful prototypes for an eventual standard.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, COMPUTER PERFORMANCE EVALUATION: REPORT OF THE 1973 NBS/ACM WORKSHOP, MARCH 27–30, 1973, SAN DIEGO, CALIF. (Issued August 1975)

Standards in Performance Evaluation and Measurement

R. W. Bemer

Honeywell Information Systems, Phoenix, Arizona 85005

Giving "evaluation" equal billing with "measurement" opens the door to discussion of performance that is good or bad, as opposed to fast or slow. Through this opening come considerations of security and confidentiality, validation of software and hardware means for performing arithmetic operations and evaluating mathematical functions (to varying degrees of precision and accuracy), code independency, auditing and warranty, optional optimization in compilation of running programs in high-level languages, and retention of statistics of every aspect of operation—for later analysis and reduction of duplicate work.

Key words: Accuracy; audit; certification; code-independent; documentation; optimization; precision; run statistics; security; terminology; validation; warranty.

1. Justification

The United States Government has imposed certain requirements upon the manufacture of automobiles, i.e., to be constructed so as to withstand collision at X kph without sustaining more than $Y in damage, or the like. The Government has stated that requiring such action is within its right to protect the safety of its citizens.

Perhaps the reason that analogy of automobiles to computers is so facile is that computers are also a major restructurer of society. The newer computer uses have a greater than ever proportion of integration into human activities (even into the automobile). It seems certain that the computer has a direct effect upon not only the safety of our citizens, but also upon other rights. It might thus be reasonable to demand that software and hardware should also be built to certain standards to protect these rights.

Giving "evaluation" equal billing with "measurement" in the discussion of performance of computer systems is a major step, for it permits us to subsume good and bad performance as well as fast and slow performance. It enables us to view the need for confidentiality and security concurrently with performance measurement. There is probably much commonality in the requirements for both.

2. Nomenclature

The present intense efforts on performance evaluation and measurement indicate a movement toward professionalism in the computing field. Yet inspection of successful professions shows the basic need for standard nomenclature, and this is lacking in our field. In particular, the American National Standard Vocabulary is to be renamed as a dictionary; this is quite proper, for it is only a list of defined usage in alphabetical order of the terms. It has no structure, whereas the IFIP/ICC Vocabulary did. Imagine a dictionary for the botanist!

And did you ever see such a sloppy term as "overhead"?

We might start with the primitive of:

Work—Answer-producing
—Answer-validating
Not Work—Scheduling
—Monitoring
—Allocating Resources
—Reporting
—etc.

Another partitioning includes people as well—in a time sequence of software preparation, testing and validation, production runs, and modification. All of these need to have subactivities named and defined more rigorously than at present. The jargon of JCL is incomprehensible to those that use other systems, and vice versa in many cases.

We need standard terminology for the operating system functions—resource management, data management, core compaction, incomplete allocation attempts, waiting, swapping, saving for restart or protection against crash, user validation, etc., etc., so that the smaller functions and program kernels can be assigned to their proper place in the classification structure.

These are the working functions, which would go on whether or not the performance was measured. Similarly, we need good definitions of the monitoring and measuring functions.

3. Reporting

A distinction should be made between the two types of reporting—online for operator intervention and change, and offline (later) for accounting and analysis. Both provide opportunities for performance improvement. The most improvement is likely to be available through providing the operator with sufficient tools, once the operating system has been shaken down somewhat. (I would prefer to see operators of higher caliber than programmers, at least for complex systems, with this reflected in the promotion scale.)

ANSI X3 is very unlikely to achieve a standard for operating systems. There could be some standardization in the subset of reporting activities and their appearance to operators. This might seem unnecessary in the present situation, where programmers change installations with a basic knowledge of some standard programming language, whereas operators scarcely ever do so. But wait until management finds out that some operators have skills, and a feel for tuning a system, that make them far more valuable than any programmer who knows COBOL only.

Accordingly, it is not too early to seek some standards for reporting, by both printed message and analog displays, of resources allocated and used with respect to the individual jobs or batches of jobs. From the crude manometer display on up, more than resource consumption must be reported; contention must also be reported and identified to specific tasks, i.e., resource wastage as well as resource consumption.

4. Software Construction

4.1. Code Independency

All software, whether it be written in high-level or assembly language, should be code-independent from the native character code of the CPU and/or any other code such as the ISO Code (ASCII) and EBCDIC.

The importance of this condition may be judged by the fact that the original 360 software, written without control over such code dependencies, has never been able to be converted to run the 360 as an ASCII-based machine—a feat that the hardware is fully capable of doing.

It may also be judged by an example program in the benchmark tests for the WWMCCS procurement. The source program, although written in COBOL, utilized conditional statements that were operative based upon knowledge of the collating sequence of the EBCDIC (in order to provide these benchmark programs, they were first written for the IBM 360/50, and so tested). The HIS 6000 programmers assumed from the terms of the specifications that ASCII was to be used throughout, and at first could not get correct answers. When a subroutine was inserted to mimic the EBCDIC sequence, there was an 8 percent penalty in running time.

The class of statements that can operate improperly due to code dependency is definable. Source programs may be searched mechanically (by program) for such occurrences, and offending statements at least printed out for manual inspection, if not automatic.

Alternatively, input data to program testing should be given in up to three codes—ASCII, EBCDIC, and the native CPU code if it differs. Such testing should all fall under the Quality Assurance function.

As to public warranty, all software should be certified to auditors, and in advertising, satisfactorily tested for code independency, whenever there is any possibility of portability.

4.2. Frequency of Usage

Software should be so constructed that a frequency count of execution is obtainable, upon demand, for all components. This requires a standard way of identifying such components, and conformance to standards for call and linkage (in hierarchical form, by function).

There should also be provision for count of actual machine instructions during execution of a working program (for the program itself, however, distinct from the operating system, which should have its own count). This provides a "signature" analysis of generated code. In the WWMCCS procurement, a high frequency of single-character moves indicated improper generation of object code. Rewrite resulted in a great improvement in running time.

Frequency of program component execution is quite a different thing from frequency of instruction usage. Both are useful. The latter may be accomplished satisfactorily in a Monte Carlo sense by trapping the instruction in operation at fixed intervals of time. In 600 FORTRAN, this showed that a 4-instruction linkage took up 7 percent of all running time during compilation. Two instructions were cut easily, thus improving 3.5 percent. Over the lifetime of the system, this amounts to several million dollars.

4.3. Computational Accuracy

Results, or answers, are commonly not as accurate as the programmer expects them to be. This is often due to successive operations, truncation, roundoff, basic precision used for both fixed and floating point operations. Use of greater precision should be not only under the control of the programmer, but also as a handle to the operating system. It is conceivable that the programmer should be required to state a value of expected or required accuracy for answers from a computational program segment. The operating system could randomly switch to multiple precision and rerun that segment, with an error message if the difference from the single precision answers exceeds the stated bound.

There should be a standard for floating point computation (in either hardware, firmware, or software) that says: When addition or subtraction of two floating point numbers results in an effective zero because they are of equal magnitude to the precision used, the result shall have a fixed point part of zero, with an exponent part diminished only by the precision of the fixed point part—the exponent shall not be the minimum representable. For old CPU's that do not operate in this manner, all such computations should be interrupted for logging and/or notice to the operator/programmer.

There are many studies in the literature (and the number is accelerating) that show inaccuracies in the common mathematical and business functions that exceed by far the inaccuracies in the normal arithmetic functions. This calls for certification of such functions for specific accuracy within a specific range, with public notice given—for either free or product software, arithmetic, mathematical, or business.

There should be a standard for such programmed function that requires the accuracy, execution time, and storage use to be integral with the function. Then the programmer could call for certain accuracies for general computation, and one of multiple forms for a specific function could be selected to meet (but not overmeet) that requirement.

4.4. The Compilation Process

We take the premise that programs of any significance will be compiled many times prior to successful operation, and many times later for update and modification, and that this process will move to the jurisdiction of other than the originating programmer.

Optimization is often a substantial component of running time, sometimes up to half. Therefore compilers should be constructed so that optimization is selectable.

Virtual storage or not, breaking up a large program into several components for compilation and testing is still good practice.

The compiler should have facility to flag identifiers of fewer than enough characters to make good documentation for other users. Uniqueness is not enough.

Compilers should always produce an updated source program! This should contain at least:

- An imprimatur identifying the compiler used, language features required (or not used), level, and time.
- A statement of the facilities and resources used, running time (either demanded or assigned), etc., for later analysis.
- A concordance of identifiers and statement types used (this may be in hard copy at option).
- A reblocked source program, indented to show nested levels.
- Appended list of mistake messages, if any, or an indicator of successful compilation, as far as the compiler can tell.

5. Documentation

All data on media should be self-descriptive as to format and content, regardless of whether or not it is to be used for interchange. Present labeling standards are insufficient.

It is presently difficult to associate program documentation and run instructions with the program itself, because many programs are kept in punch card form. However, with the full-scale advent of cassettes this condition should be mandatory.

Local documentation, i.e., that associated with the individual operating statements or groups of statements, may be subject to a certain minimum amount of verbiage, else the program may not pass Quality Assurance.

6. Hardware

It is difficult to make many standards for hardware design, for the technology is at a time when virtually anything is possible at a reasonable price, due to microprogramming and chips.

One definite requirement is that all CPU's should have at least two clocks—one continuous and one resettable—both fully available to software.

---

3. Workshop Discussion

Many participants felt that "standards" could not be set because performance evaluation ideas have not matured adequately. Instead, the term "guideline" was adopted by most people. One of the areas for potential guidelines was accounting data.

Browne: It should be possible to have some guidelines, even if not standards, saying that all systems shall put out the following things on an accounting basis. If it's done right, there should be some minimum guidelines for main-frame vendors and software vendors that solve some of our problems. I think this is a "must." I think we should put some guidelines down suggesting that this is a minimum kind of thing that we ought to be looking for; we'll do better later.

Bell: It seems that the epitome of what we're stranded for is for accounting data, when the systems collect essentially the same data and put it in different formats with slightly different definitions. It's apparently trivially easy to make them coincident. They ought to be coincident so that things can be done in a consistent manner. It's like having tape drives with different size reels.

Browne: There are two points to the problem. They should be receptacles for linear transformation and be consistent.

Bell: I second it.

Boehm: Ok, would somebody state precisely what it is that we're saying ought to be "musts."

Browne: I think we should write some guidelines—we must write some guidelines for minimum content in the accounting system and for a common format for accounting data.

While the need for such guidelines was clear, potential problems were noted by other participants.

Kolence: I'd like to recommend two points that I think are important. One is that along with the type and format of data of be obtained, the capability for the user of such data to obtain other new data is important. In other words, I don't think we could expect our suggestions to serve a fixed set of data that's going to be given for everything. I think it's imperative that we make a resolution open ended to permit other types of data to be collected. In other words, the facilities must be there to collect other data than what we anticipate now. That's point one. Point two is what we were talking about earlier: That integrated instrumentation systems include a minimum set of accounting data and report it well.

Jeffery: You want also to be absolutely sure that what goes into a guideline can use results from a research environment.

C. MONITOR REGISTER

1. Workshop Results

Connecting a hardware monitor to a computer system sometimes creates problems. Probe point identifications are sometimes difficult to obtain, and sometimes they do not exist. Attachment to the wrong point is easy, but detecting the problem from the resultant data is difficult. Attached probes can load circuits and cause temporary hardware malfunctioning; careless attachment can physically damage the computer. Laying cables disrupts operations as floor panels are lifted, and careful analysts often demand that computing be halted while the actual connection is performed. All this would be unnecessary if a standard attachment plug were provided.

Hardware monitors are capable of cheaply collecting data that current software monitors cannot collect easily or at all. They facilitate measurements that are independent of the host machine and therefore can be used in situations where reliability is low. In addition, these monitors can be used on different hardware and software systems so that comparisons can be made through time and across installations. Finally, communication to the machine for purposes of on-line performance enhancement is virtually impossible without some special interfacing device. While measurement through hardware is far from constituting the entire realm of performance analysis, it is important enough that mainframe vendors (and peripheral manufacturers) should recognize the user's need in this area.

The panel which met on this topic suggested that a special register be implemented for monitoring. This monitor register would be implemented differently for different hardware, and a manufacturer might choose to implement successively higher levels of the register over the lowest, level one, register. The various levels are as follows:

Level One: Lowest level register, designed to facilitate current techniques. It would consist of buffered lines to show device activity status and would have complete documentation on logical and electrical characteristics.

Level Two: A register to enable software in the host machine to communicate with the hardware monitor. It would consist of a register one word wide, loadable by the host machine's software, with half loadable from a protected state and half from an unprotected state.

Level Two (Extended): Intended to ease monitor design. It would save the unprotected half of the above mentioned word so that bit status set by a user could be maintained for that user.

Level Three: Full memory bus capability. This level would bring out (buffered) instruction address register(s), data address field(s), operation code(s), comparator status, etc.

Level Four: Communication to host system. This level would consist of a register readable (in both protected and unprotected states) by the host machine for input of special resource-allocation messages from a monitor.

The monitor register suggestion generated much discussion with some people maintaining that it assumes the current situation as the long-term technological environment. For example, micro-programmable devices throughout a system might make definition of words like "device active" impossible. Therefore future monitoring capabilities should be designed by manufacturers so as to provide a recommended set of data, but with complete freedom of choice as to the technology and architecture of those capabilities. Other participants argued that the results of past vendor designs for facilitating performance analyses did not indicate that users should wait to see what vendors might implement. Attempts to obtain a definitive consensus on this issue by vote were inconclusive, with many abstentions.

2. Workshop Discussion

Several performance improvement practitioners noted their enthusiasm for having a well-defined data collection facility integrated in a computer's hardware/software. They felt that a monitor register was the most appropriate manner for obtaining data from this facility. Other participants, however, felt that other considerations made such a technique unreasonable.

Morrison: I just feel that you're not going to get what you're asking for by asking for so much. It was a wonderful idea 8 years ago. You have heard that view expressed to this particular group. The concept of integrated monitoring has been intermingled with hardware monitoring and current practices of deter-

mining component utilizations. It's just not going to be that way in the future. The words about implementing an integrated performance reporting facility are what I think is the essential idea.

Boehm: Could you name a specific area where you think a monitor register would cut into the freedom of a vendor?

Morrison: Well, the first issue is how much goes into every machine versus how much the particular customer wants. That is not the primary issue involved. It's there, but that's not the main issue. The problem involves a lot of other things: training of the people who are going to have to maintain these more complex machines—all sorts of things to tell them, what not to tell them. The whole area of what goes into a computer—including those ideas that just add to cost—the idea of how a manufacturer examines a number of new concepts for incorporation. Out of well over a hundred well thought-out, justified, quantified ideas come only a few which should add to the cost of the new machines.

Second, much of the separate stuff is coming into every new machine so that you can get high performance. Today we think about an operating system as not included in the hardware but it may happen. I do feel that you're possibly locking out something when you attempt to consider specific hardware implementations. With more integration you're going to have less capability to control variables. If you started with the guidelines and said, "Let's generate information and have some control over what we're going to do," I could understand that.

Wilner: I think that if you're talking about writing a guideline, I feel capable of interpreting whatever words are in it in terms of new architecture.

Boehm: I'd like to hear a technical discussion between Bob and Wayne as far as what the fundamental technical difference is between what they're saying. Wayne is saying that any guideline he has can be incorporated. It is sufficiently lucidly worded as far as he's concerned so that he can incorporate it in there. But you don't feel that.

Morrison: No, oh no.

Warner: I think the principal difference is not a technical consideration, but a marketing consideration. I just wanted to say that and no comment is necessary.

Boehm: Bob Johnson, could we get your reaction to this from a corporate management standpoint?

Johnson: The problem I have is being worried about what's practical and realizable. Most of you talk about finding in your logic manuals by dint of great effort a combination of some probe points with real meaning. At least a number of our systems people and development people are finding that they can't find some of these things. Whether or not they can be found, I don't know. I haven't gone into it myself. I don't think these problems are because of reluctance or inhibition or any of that; it's just not there in a form that we could really depend on. Another aspect that I think important for us to consider is that these things that we're asking for are things we today think are important. We don't know that they are. They allow us to do something. But in support of Bob here, the things that may be important to us 3 to 5 years from now may not at all be the things that you're asking about. If we're going to pick an example, take the number of instructions executed. If you're going to make applications processing machines and execute the application directly, the definition of an instruction isn't clear and this whole discussion gets very tenuous. What is it we're looking for? Perhaps, we're executing all types of language statements and this execution is going on in many different places in the machine at once. What is it that's going on in the simple instruction? There isn't any way to track it.

Kiviat: You don't know what the machines are like that we're going to have. You can't tell us what you might want to measure for them, and we all know it's too late to do anything about the machines that we do know something about. So let's stop what seems to be a fruitless discussion.

D. WORKLOAD CHARACTERIZATION

1. Importance

A recurrent topic during the Workshop was the necessity for better means of workload characterization, i.e., of determining meaningful categories of workload types, and parametric forms for describing a workload of a given type. Such a capability is important because it provides the necessary framework for:

1. Verifying performance improvement hypotheses by enabling an analyst to normalize performance

improvements during periods of changing workload;
2. Predicting trends in computer usage of various types, and predicting the resulting resource strains;
3. Providing functional forms and parametric data to enhance analytic modeling;
4. Providing useful parameters for closed-loop monitor-scheduler modules in advanced operating systems;
5. Improving the quality and comparability of benchmaking activities.

2. Complexity of the Workload Characterization Problem

As mentioned above, a working group was convened at the Workshop to try to develop a definitive workload characterization. The group found it was not all that easy. As far as categorization, there are several classes of categories which include at least the following:

1. Job or transaction characteristics
   a. Value of job completion (function of time, input data, state of application system, etc.)
   b. Resource demands
      1. By component
         (a) hardware (CPU, core, disk, channel, etc.)
         (b) software (compiler, I/O, user code, etc.)
      2. By usage pattern (probability distributions of resource demands)
         (a) by timing
         (b) by amount
2. Inter-job characteristics
   a. Dependence of job completion value on completion of other jobs
   b. Probability distributions of job interarrival times

The process of determining workload types appears to involve an intuitive cluster analysis with respect to the above categories, in order to identify clusters of jobs with similar characteristics such as student jobs, accounting jobs, I/O-bound jobs, on-line transactions, etc. Determining the appropriate parametric forms for each type generally involves a similar, but usually more quantitative analysis. Some guidelines with respect to these determinations are given in the conclusions below.

3. Conclusions

1. The most useful form and level of detail of a workload characterization depends on its application. This implies that workload characterization is generally an iterative, circular process.
2. A workload characterization is useful only to the extent that the necessary parametric information is easily available. A good example is the two-parameter job characterization (CPU seconds of execution and kilobyte-minutes of core residence) sufficient to provide effective job scheduling in the Air Force business data processing system cited in Section I.B.4. A counter example would be a 100×100 contingent probability table of memory references in a complex Monte Carlo simulation model.
3. Workload characterizations are often machine-dependent. For example, initiator and terminator activities are quite time consuming in IBM 360 machines, but usually negligible on CDC equipment. This implies the need for extreme caution when characterizing a workload to serve as a reference for benchmark tests during the equipment selection and procurement process. In some cases, a workload characterization suitable for benchmarking may be unachievable.
4. The primary needs in the workload characterization area are for an increased level of empirical information exchange on the utility and achievability of various characterizations. and for further complementary work toward an underlying theory which is relevant and accurate both in explaining previous situations and in predicting future situations.

4. Workshop Discussion

A good deal of the discussion focused on the machine-dependence of workload characterization and its implications for benchmarking as a means of rating alternative computers in procurement evaluation.

Boehm: Our panel last night felt that this machine-dependence makes a major caveat for anybody who is trying to use workload characterization for benchmarks. Unless you realize that, you can get into very

big trouble taking something that was a good workload characterization on one machine and using that characterization to benchmark another machine.

Wilner: What's a big caveat. Something that's likely to fail?

Boehm: Not necessarily. Our feeling was that the most productive thing to do was to consider the adequacy of the workload characterization not as a given, but as a hypothesis. Then you can be prepared to test whether that hypothesis is true, and, if you find out that your workload characterization isn't any good on another machine, then you know that you've got to do something else to evaluate the relative performances.

Wilner: Wouldn't it be a simpler course to avoid thinking about benchmarking for the present time?

Boehm: But people still have to choose computing machines. If you do that independently of any characterization of your workload, you'll be worse off than starting somewhere and being sensible about it.

Wilner: How sensible is it to do something that's so machine dependent? How does anyone know what his comparisons mean?

Boehm: What's the alternative?

Bell: Don't buy any more machines for a while.

Wilner: No, no, no!

Kimbleton: Workload characterization can also differ on benchmarks depending on whether you make the assumption that each machine runs exactly the same code, on the one hand, or that each one solves the problem the best way it can, on the other hand.

Nielsen: One way we looked at this (which isn't a solution, but was our way of characterizing it) is that you can either take a job-oriented or a function-oriented workload characterization. On the former, you would select a set of jobs and hope that that characterization holds true on the other machines you're working on. On the latter, you can take the approach of trying to characterize the properties in the workload which should be compared. That is, you're saying what the important characteristics by type are independent of job load, and then trying to match those characteristics on what you're doing statistically.

Other portions of the discussion focused on the interdependence of a workload characterization and what the system is being used for, and on the general unpredictability of an installation's workload characteristics.

Schwetman: Another point was that the form or the format of the characterization is dictated by what you're going to use it for. For example, if Dick Muntz wanted some data on workloads for an analytic model, he's got one set of things he really needs. If Norm Nielsen wants to draw up a model for a simulator, he might want different things, such as more detail in sequencing, or more information for scheduling jobs or for fine tuning.

Nielsen: This implies two things. One is that what you use to characterize a workload is a function of what you want to do with it, in that we can always come up with a counter example, for which a given workload feature is irrelevant or inappropriate. The other is that at the present time, we don't think that there is a theory that can say: "This is the thing that characterizes the workload." There's so much that is empirical. The things that are important are a function of what you want to do and the machines that you have—the different machines, the different operating systems and different things that cause changes in the quality of the workload. You have to experiment with your system to find the things about workload that have an effect on my system.

Schwetman: Another point we discussed was that you really can't prove anything about the future. Computer systems are no better or worse in that respect than any other field. Thus, you really shouldn't expect that a model would accurately predict the future. All you can say is that you've known cases where in the past it worked, and the same thing goes for all the other techniques you can think of, in economic modeling, or simulation or anything. And so my comment was, you can also not predict the future success of a marriage necessarily. All you can do is make a good guess, and sometimes you're right. Sometimes things work out and occasionally it fails. And that's the point. In spite of the fact of the uncertainty, people still do get married. And the reward outweighs the risk, I guess.

E. PROFESSIONAL SOCIETY ACTIVITIES

1. Workshop Results

Workshop participants voted 23 to 2 in favor of the following recommendation:

> The professional societies should treat this field no differently than any other. The societies should provide the usual channels of communication but should not themselves try to provide or measure compliance to standards.

The dissenters pointed out that there exist some professional societies which promote and monitor standards, and that professional societies in the computer field should exert more leadership in this direction. The majority opinion was that other types of organization (e.g., NBS, ANSI, CBEMA) were better suited for standards roles, and further that the major needs in the CPE field at this point were along the traditional professional society lines of stimulating and facilitating professional communication and information dissemination.

In the area of information dissemination, existing publication channels were considered generally adequate, as long as SHARE and ACM's Special Interest Groups on Measurement and Evaluation (SIGMETRICS) and Simulation (SIGSIM) continue their trend toward publication of well-documented results of CPE efforts. There was some concern that professional journals in the computing field were overly biased toward publishing theoretical rather than empirical results.

One topic of particular concern was that of model validation. Users of analytic or simulation models of computer systems currently have no way of determining the extent to which the model has been validated for their situation, often leading to lost time and effort, to duplicative validation activities, and at times to inappropriate management decisions. Workshop participants felt that much could be done, within professional societies and elsewhere, to encourage and communicate the results of model validation activities. The initiatives at the February 1973 ACM SIGMETRICS Conference were a valuable first step in this direction.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, Computer Performance Evaluation: Report of the 1973 NBS/ACM Workshop, March 27–30, 1973, San Dieco, Calif. (Issued August 1975)

The Role of the Technical Societies in the Field of Computer Measurement

R. W. Hamming

Bell Laboratories, Murray Hills, N.J. 07974

It is comparatively easy to make measurements of computer performance, but this does not mean that there is, or can be, any single set of "right" measurements of performance—much as we may wish otherwise! This being the situation, the Technical Societies should not get themselves involved in trying to set standards of measurement (in the sense of what to measure), though they should encourage high quality measurement and subsequent data processing, publication, and oral presentation of results.

Key words: Computer measurement; technical societies.

Since the role that the technical societies should play in the field of measuring computer performance is very broad and somewhat delicate it is necessary to approach the topic from a philosophical point of view, and also to note that this is not a scientific paper but rather is merely opinions.

As a background I observe that we live in an age of social and political experimentation. The past ten years, and more, have seen a great many very costly social experiments tried, most of which have failed to live up to the test of reasonable results, and many of which have left us worse off than we were before we started. Thus a healthy degree of cynicism is justified when faced with proposals for a Computer Society activity in the area of computer performance measurement, especially since once beyond the lowest level of measuring what happened there immediately appear various consequences and goals that fall close to, or even into, the areas of social and political activities.

Our American society has apparently finally realized that we have limited resources and cannot do everything. The Federal Government appears to be embarked on a path of retrenchment, different from the immediate past when every new proposal had a good chance of getting money. Similarly, the technical societies have found that they too have limited resources and are gradually cutting back various "well intentioned" activities in order to get a balanced budget—and possibly even reduce the outstanding indebtedness.

With this as a background, let me move a bit forward toward the main topic, my view of the role of the Technical Societies. Many of the proposals that are made are viewed against the background of successful ventures in the hard, physical sciences, such as the atomic bomb and the moon shots. Usually the problems that we pose in the physical sciences have solutions with known characteristics which can be used both to guide us in the search for a solution and to measure the solution when we think we have found one. But the meaning of a solution in the social sciences is not so clear, nor is the very existence of a solution guaranteed—no matter how much we wish there were one! For example, what would be a solution to the current Israeli-Arab situation? Do you believe that there exists anything that would be widely accepted as a solution to the problem? Thus, before we propose to measure computers are we sure that there are measures that would be widely accepted as relevant? It would be nice if there were, but past experience, running over more than 20 years, shows that we have never come to any agreement on how to rate computers. Not that we have not repeatedly tried! In the scientific application area it has been regularly proposed that a set of test problems be found for rating machines, but no such set of prob-

lems has yet emerged from all the effort that has universal consent as being relevant. If this is so in the scientific area, how much more diffcult will it be in the business area! And in other areas!

The governing rules in much of mechanics are second order differential equations so that, in principle, all we need to know are the initial positions and velocities in order to know everything that will happen in the future. The behavior is called the solution. But I doubt that in the social sciences we will ever find such simple underlying formulas, and I even doubt that there can be such simplicity. Instead, what I expect we will find are a number of complex, general rules, each somewhat vague and surrounded by a halo of exceptions that are "explainable after the fact" but are not "predictable." Thus I believe that in many areas we will in the future be in the same position that we are now; we cannot reliably predict but we can account for what happens after the fact.

Why am I talking so much about the social sciences? Simply to remind you that the physical sciences are based on the external world of observations. It is true that we have learned that to some extent we condition what we will see by the way we observe—thus in some senses the special theory of relativity revealed the external world. On the other hand the social sciences are centered around the human and his behavior, and we have found these sciences remarkably difficult to develop with the precision of the physical sciences.

When we come to the computer sciences they are mainly a creation of the human mind (this in spite of the very real physical computer), so that they are, in many respects, on the far side of the social sciences from the physical sciences. They will be even softer than the social sciences.

I hope that you now see why I take a dim view of much of the past measurement in the computer sciences. I feel that those measurements which concentrate on the machine itself will have many of the attributes of the physical science measurements, but as you move away and begin to include software you move onto less firm ground.

It is when you begin to recognize that there is a human as well as a machine, and it is their interaction that is important, then you get well into the social sciences and their softness, vagueness, and trouble in getting at fundamental results of lasting importance. Beyond the human-machine interaction is the even more basic point that both the humans and the machine are financed to meet corporate and institutional goals; then you are at the foundations of why the whole exists.

Finally, when you try to measure the ideas we have in computer science you find that you have little precedence for guidance. Because, to me at least, much of computer science is concerned with human ideas. We need to try to measure, but should not expect immediate, practical results.

Thus in the field of computer measurement we face the dilemma, that which we can measure very easily is only slightly relevant to the purposes of the computer and the people running it, including the programmer, the planners, etc. What we want to measure is the effectiveness of the whole system, and we have so far made only the slightest efforts in this direction. Further, the scientific aspects of computer science are even more elusive.

The response of most of the audience is, I suspect, "Let us ignore what we cannot measure and get on with what we can measure." But we are concerned with a whole system, and it is well known that the optimization of a part of a complex system is usually detrimental to the performance of the whole. The current concentration on machine optimization is often, I suspect, done at the cost of the total system performance. Although improvements are made on paper in the whole system, the "improvements" are apt to be counterproductive.

It is for these reasons that I am reluctant to see the computer societies get into the difficult field of measurement. We are likely to do more harm than good in the long run if we try to set up standards in the areas where we can measure easily and voluminously!—and thereby neglect the measurement of the larger goals which we do not know how to do.

With all this negative talk, let me say a little on the positive side. The societies can, should, and probably will, encourage meetings for the exchange of information, ideas, and techniques, as well as mutual encouragement to persist. And the societies will probably provide reasonable space for publication of the better results. But let me make myself perfectly clear (as the saying goes) I do not think that the technical societies have the resources in ideas, money, or manpower to measure computers directly, or even to propose worthwhile measures much beyond what we are already doing.

I further believe that the technical societies should not try to compete with established commercial efforts. While this is not an exclusive rule, I feel that the

technical societies should concentrate on the more scientific aspects of their fields and avoid the more engineering aspects. With all the measurements that have been made in the past I have yet to see much of scientific merit emerge—what has emerged is a lot of very valuable techniques for making measurements. I would like to believe that we can gradually create a science of measurements and their processing and interpretation, but I am not certain that it will happen in the near future.

All this should not be interpreted to mean that I am against measurement. I have a motto:

> "Without measurement it is difficult to have a science."

which should indicate that I feel otherwise. What I am opposed to is the technical societies getting directly into the act. Their proper role is one of helping and encouraging their members to do good scientific work.

3. Workshop Discussion

Some participants felt that professional societies should take a much more active role in such areas as standards; others felt this could easily lead to abuses and discouragement of diversity. For example, here are some comments on the panel's proposed resolution.

> The professional society panel believes that professional societies should treat the field no differently than any other. The societies provide the usual channels of communication, but should not themselves try to provide or measure compliance to standards.

Morrison: I disagree that that is a correct assessment of what some professional societies do. In the area of acoustical standards, for example, the professional societies most certainly promulgate, in regard to standardization of terms.

Kimbleton: The danger is that a situation can evolve in which a small group of individuals in, say, ACM, have a lot of control over the disposition of ACM's position and related funds. The group can then imply that it receives and disperses ACM approval for the direction and scope of the scientific effort, giving its position an unfair advantage. Particularly since CPE is a field in a great deal of flux, it does not seem like a professional society like ACM should come out with a position paper on the field. It should serve as an enabling means of discussion and clarification of issues in the field.

Morrison: But the point is, having promulgated a position, a society can get people working on the issues. Under some conditions, they can change their positions, but if you take no position you are not involved.

F. RESEARCH AND DEVELOPMENT

1. Workshop Results

Workshop participants were strongly divided on the matter of R&D priorities in the CPE field. After some discussion, it appeared that the most productive approach would be to ask participants to list their choices of the (roughly) three most important R&D projects they would fund if in a position to do so. The results, representing 23 responses, are given in Table 5 below which is reproduced from Chapter 1 for clarity.

TABLE 5.—*Desired R&D projects*

Theory (4 categories)	8[a]
Measures and Criteria	7
Model validation and refinement	7
Workload characterization	6
National CPE laboratory	5
Closed-loop monitor-manager	5
Representation of systems and information structures	5
Comparative measurements collection	4
Hardware-software monitor	2
Variability and predictability	2
Statistical methods	1
Programmer productivity determinates	1

[a]General theory, 3; Analytical models, 3; Queueing theory, 1; Work-energy theory, 1.

Most of the entries are fairly self-explanatory, but the national CPE laboratory deserves some added explanation. It would involve the provision of a computing center stocked with measurement and evaluation tools available to theorists and experimenters wishing to test theories and hypotheses on computer system performance.

Most of the discussion of this concept centered on the problem of maintaining a representative real-world workload on an experimental system. Many users with deadlines would prefer not to use such a system even if it were available free of charge. However, it would be most valuable for the facility to run live production work, both by itself and in concert with a set of representative parameterized workload characterizations.

Other recommendations of the R&D panel drawing more general support from the Workshop participants were various information dissemination and consolidation activities such as specialized workshops and conferences on CPE theory, workload characterization, etc., channels for review and evaluation of R&D work, and reference books for the field. In this last area, participants were polled for their opinions on the most valuable yet-unwritten document in the CPE field. The results for the nine responses received are given in Table 6.

TABLE 6.—*Valuable documents: Unwritten*

4—Measurement & Evaluation Handbook (when, how to use accounting data, monitors, simulators, etc.)

1—Exposé of the Unreliability of Every Known Comparative Measure

1—Facilities Management for Small-to-Medium Computer Centers

1—Organizing and Managing Measurement and Evaluation

1—Applied Statistics for Computer Performance Analysts

1—Integration of Four Types of Performance Measurement into a Single Tool (internal hardware monitors, external hardware monitors, software monitor, "mini" software monitor).

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, Computer Performance Evaluation: Report of the 1973 NBS/ACM Workshop, March 27–30, 1973, San Diego, Calif. (Issued August 1975)

Computer Performance Evaluation—R&D

James H. Burrows

Headquarters, USAF, Washington, D.C. 20330

Computer Performance Evaluation has moved to the forefront of a long list of tools to help the field practitioner. However, as long as R&D is applied only to large installations pushing the margin of feasibility (a noble and rewarding effort) that R&D will not contribute to the majority of data processing installations. Something useful is needed for the more normal installation.

In addition, it is becoming increasingly clear that CPE leads one to discuss and evaluate procedures and goals set for and acting upon the whole "user" community and not just the hardware monitor. More needs to be done in the external environment.

Key words: Accounting; computer performance evaluation; efficiency; measurement; research and development.

I would like to bring emphasis to the use of products of R&D in this area. Funds will not be made available nor will the potential social benefits of the R&D work be achieved if R&D is not addressed to "real" problems and to "real" solutions.

By "real" solutions, I mean solutions that can be taken advantage of by the appropriate "real" people —people with limited or constrained resources and with loads that are pushing those constraints.

I take this view, also, because I feel the product follows the market more than the product making the market when it comes to CPE. Simple techniques must be available to the normal installation to allow the most benefits to accrue. Large installations, university installations and multiple similar installations probably have already adopted many of the techniques of which we talk.

In point of fact, in my limited experience, these installation types are beginning to see new dimensions to the CPE problem. Many of them are finding that much of the remaining freedom to achieve further benefits does not remain in the center's area of jurisdiction but resides with, for example:

a. The external customer, his requirements and his programmers;
b. The bookkeepers who need simple, "fair," charging algorithms which usually charge on the "average" although decisions usually cost on the margin;
c. Management decisions as to type, quantity and cost of service to be supplied to various and varying customer types.

The computer center should not be allowed to legislate on its use in such a way as to displace costs from the center and force them back on the customer in some other account, such as his salary account. However, this does occur.

On-line checkout is possibly an inefficient hardware use, and it increases the costs and complicates the management procedures for the center. So! But contention has it that the value of on-line operation varies significantly over the spectrum of programmer talents. Do we expect to expand CPE to include these areas? If not, we do not answer the "real" world problem; but to do so would take us further over our heads.

I say we must talk to these "real" world management problems or we lose much of our market. The question is how?

It is not the case that the majority in count or

dollar expenditure form in the DP community is pursuing applications that are on the margin or edge of technical feasibility. However, many of our studies and results have been made in such an arena.

Can we, out of these highly technical endeavors, come up with techniques, guidelines, etc., for the other segments of the market? Do they need tools as sophisticated, as precise? Are there general guidelines, and can we validate them? Have we?

I have prepared a chart of possible times for the use of CPE efforts. (See Figure 1)

	Before Software Design	During Implementation	After Initial Operations
Before equipment selection &installation	A	C	E
After equipment installation	B	D	F

Figure 1

Whether some of these boxes, such as E, are vacuous or not can be argued depending on your belief in "software first" concepts. Independent of our leanings, it is clear to us all that most of the market is in state F.

In addition, I have a list of people who are involved in our world:

Computer Center	Users	Management
Operators	The "real" user	Auditors
Schedulers	Requirement Analysts	Accountants
Systems	Program Designers	Operating
Hardware	(Strategy)	Managers
Managers	Program Implementers	Resource
	(Tactics)	Managers

What do we have to say to these people? Are we talking to all of them? Should we be?

I have left out of the list stated above many of our colleagues. This is not to disparage their work nor to minimize their contributions. For example, the extensive work effort put in by IBM to finalize the design of the "cache" for the 360/85 staggers the mind in both completeness and probable cost. Most of this type of practitioner is amply supplied with both talent and money. I feel we must address the "poor" people of which we seem to have an abundance.

I know that it is easier to sell "good" advice to a technically competent buyer, and it is almost as easy to sell "any" advice to the technically naive as it is to sell "good" advice. In fact, "any" sellers almost always "oversell"; "good" sellers undersell—a penalty in the average market place.

Thus, most of us concentrate on helping the technically competent who knows how to use us rather than being missionaries among the naive. How can we turn this around? I feel that NBS and the ACM can help in developing and selling the "good" brand advice, and I hope that is their goal.

Who among us is collecting the questions that practical (and poor) managers must answer in the course of managing their installations? How can they use any guiding principles we may promulgate, and how can they use current or new PME data to help them classify their condition and develop and select among alternatives?

Has anyone done any studies of whether "homogeneous" programs, all of similar characteristics, can be better run than highly variable? Most of us avoid the extremes; should we?

Is a standard core program module size of which multiples can be chosen more efficient than varying sizes? How do we measure efficiency—at the scheduler's elbow or at the programmer's?

Can we develop a post-run analysis program that would point out "what could have been done" versus what was done? How could it be used? By whom?

Who should be responsible for discovering, by way of analysis of data specially collected during a run, that local or "tactical" changes can speed up processing? This probably depends on what tools are available and what special knowledge is needed to use them. Can we reduce this special knowledge to a minimum?

Are there any "runtime" tools to point out strategic possibilities for improvement in programs or sets of associated programs?

What kind of "charging" algorithms have the effects alluded to by Watson [1] of providing the "right" incentives?

[1] R-573-NASA/PR—Computer Performance Analysis: Applications of Accounting Data, R. I. Watson, May 1971, Rand, page 53.

Can appropriate incentives be worked out for all the participants in system creation and operation? Do they vary and how?

I hope our discussion at the workshop covers things for the average installation as well as for the expert. Many of us need more light on where we are walking today than on the pinnacles of the future.

3. Workshop Discussion

Much of the discussion centered around the concept of a national CPE laboratory, which would give CPE researchers access to large machines and representative workloads for experimental purposes.

Burrows: Another valuable R&D activity would be to provide for both the theorist and pragmatist a computational arena for testing their ideas. It doesn't have to have standardized workloads, but the workload ought to be well instrumented and checklisted, so that what's flowing through the arena is well-defined and available. And people could go there and check out both the uses of a concept and its theory as well.

Browne: To be useful the workload in such an arena would have to be production work—above all else, production work. Without production work you can forget it.

Kolence: That was the idea here. You wouldn't necessarily have one installation and one computer and do all your work on that, but rather there would be a capability to go around to different places.

Burrows: What installation manager with a realistic workload is going to let people experiment with his live system? Does he get an extra $100,000 a year to suffer the pain of having guys come in and borrow all his stuff?

Browne: All you'd have to do is halve your service fee.

Kiviat: Not true. I had a hardware monitor that I got for free on a trial basis. I went to a number of installations and said, "Would you like me to attach it so you can see what it does?" I couldn't get any takers.

G. EDUCATION AND TRAINING

1. Workshop Results

The Workshop participants strongly endorsed the recommendations of the Education and Training panel that a coordinated program of CPE education and training should be established and supported by universities, funding agencies, large user organizations, and professional societies. The program would have two main focal points for educational leverage: education for motivation to increase awareness of the general potentials, pitfalls, and procedures of CPE; and education for competence to increase the quality and quantity of practitioners in the CPE field.

In the area of education for motivation, the main targets are:

- Managers at the intersection of authority over computing resources and computer users;
- Computer center managers;
- Lead systems programmers;
- Users;
- Vendors—hardware and software.

The most promising mechanisms for attracting and motivating the above individuals are:

- Seminars, tutorials, and workshops for focal points of purchasing control (e.g., state agency D.P. boards, professional organizations, such as the American Bankers' Association, American Management Association, and Federal agencies);
- Books and periodicals: case histories, etc.

In the education for competence area, the main targets are:

- System programmers;
- Application programmers;
- System designers;
- Direct users;
- New professionals entering the field.

The most appropriate mechanisms for attracting and motivating the above individuals are:

- University—regular students
  —continuing education
- Co-op programs

- Summer courses
- Books, periodicals
- Professional societies—ACM, IEEE, etc.

In addition more detailed recommendations for a university CPE education program are formulated by Noe in the following Section. Specifically, university education in measurements and evaluation has the opportunity to do the following:

1. Spread a performance-oriented viewpoint among those preparing for teaching and practice in the field. This should orient problem solvers to attack what is important, not just what is interesting. When resources permit it, this can be extended to professionals returning to the university for "refresher" courses.
2. Stimulate research and development of means for measurement of complex computer hardware and software systems through dissemination of understanding of the problems and possibilities.
3. Influence other computer science courses so that they include measurement and evaluation viewpoints relevant to the particular topics, such as compilers, operating systems, architecture, logical design and data structures.

Toward this end, the following recommendations were developed in the university education area:

1. Measurement and evaluation viewpoints and techniques should be taught at the university level to spread consciousness of the importance of the topics, and to encourage research and development.
2. Initially this should be taught as a separate topic, through formal courses and individual studies and projects. The ultimate aim should be toward inclusion in other courses where a measurement and evaluation view is important, and the separate course work should be needed only for advanced topics for students who specialize.
3. When taught as a separate course, measurement and evaluation should be placed at an intermediate level—after students are well aware of the function of hardware and software systems, but before advanced courses on design of such systems.
4. Familiarity with statistical methods should be acquired through prerequisites and should only have to be reviewed as part of the measurement and evaluation course work.
5. Measurement and evaluation should be taught in conjunction with course work on modeling of computer systems.
6. The particular type of modeling emphasized is less important than the viewpoint relating the modeling method to measurements and evaluation. The models (be they predictive or descriptive) should be used for their ability to provide evaluative information, and for their provision of a context for communications about measurements and their meaning.
7. Information should be exchanged on a continuing basis concerning the concepts to be taught and the most effective methods of conveying them. The SIGCSE Bulletin of the ACM provides one good forum for such exchange—probably better than the Performance Evaluation Review which is received by ACM SIGME members, who are already convinced of the topic's importance.
8. The most important recommendation is for continued research attention to develop the principles pertinent to measurement and evaluation of computers. This is a joint responsibility of those in industry, government and the universities.

In view of the general consensus at the Workshop that a great deal of the improvement available through CPE techniques could be achieved with present-day technology, a high priority on educational activities to unlock such a large savings potential appears well justified.

NATIONAL BUREAU OF STANDARDS SPECIAL PUBLICATION 406, Computer Performance Evaluation: Report of the 1973 NBS/ACM Workshop, March 27–30, 1973, San Diego, Calif. (Issued August 1975)

University Education in Computer Measurement and Evaluation

Jerre D. Noe

University of Washington, Seattle, Washington 98105

The paper presents the view that computer measurement and evaluation should be taught in universities to stimulate research activity and to establish in the minds of students the importance of the measurement and evaluation viewpoint. It is recommended that measurement and evaluation initially be taught as a separate course, but the ultimate aim should be for the viewpoint to pervade all course work on hardware and software systems, at which time the need for the specific course should vanish. A list of suggested concepts is included.

Key words: Computer measurement and evaluation; computer science curriculum; hardware monitors; modeling.

1. Introduction

Why is it important to have university-level education in measurements and evaluation? It is tempting to ignore the topic because it is not well developed and because it intersects so many well established courses that it seems difficult to determine what its relationship should be to these other parts of the computer science field. However, computer measurement and evaluation needs research and development, and there is need for dissemination of the solved and unsolved problems and of their importance to other aspects of computer science.

Specifically, university education in measurements and evaluation has the opportunity to do the following:

1. Spread a performance-oriented viewpoint among those preparing for teaching and practice in the field. This should orient problem solvers to attack what is important, not just what is interesting. When resources permit it, this can be extended to professionals returning to the university for "refresher" courses.
2. Stimulate research and development of means for measurement of complex computer hardware and software systems through dissemination of understanding of the problems and possibilities.
3. Influence other computer science courses so that they include measurement and evaluation viewpoints relevant to the particular topics, such as compilers, operating systems, architecture, logical design and data structures.

A number of experimental instructional programs are under way in various universities, but there is wide variation among the approaches. The rate of growth of interest in measurements and evaluation is very high in the computer field and this is leading to an increase in the growth of courses dealing with such topics. However, course development is difficult because the principles that are important to measurements and evaluation are diffuse and, in some cases, not established (an example being a machine-independent measure of computing work). Also, many of the viewpoints require direct practical experience, but this can easily become too involved for inclusion in a university course during a quarter or semester. Furthermore, the many techniques useable in measurements and evaluation, and the many areas where evaluation viewpoints are needed, make for a difficult decision regarding relationships with other courses.

All these points lead to the need for recommendations concerning course work, so that those just starting may draw upon the successes and failures of others

who have been teaching in the area. Among these needed recommendations are: Concepts to be taught; methods of teaching; relationship to other courses in terms of level of students and prerequisites.

2. Survey of Current Activity

Representatives of 19 universities, in addition to the author's home campus, were contacted in the search for formal and informal activity relating to measurements and evaluation. There is no claim that this is an exhaustive survey. It has been accomplished through contacts known to the author, augmented by follow-up on further suggestions from those who were contacted. The author would appreciate hearing from other active parties. Among those contacted, three [1] have specific courses dealing with computer measurements and evaluation: eleven of them have measurement and evaluation portions of other courses, although they have no single course concerned with the topic. The remaining universities have no detectable course work in the field but some do have research and development underway that trains students on an individual basis.

Many papers have been published, particularly since 1970, and there is need for an up-to-date annotated bibliography. Several efforts to provide one are reported to be underway (one being by the Los Angeles Chapter of SIGMETRICS) but are not yet published.[2] The only text devoted entirely to measurement and evaluation is one that has just been published (1973) by Prentice-Hall: "Evaluation and Measurement Techniques for Digital Computer Systems," by M. E. Drummond, Jr. Some aspects of the field are covered in the conference papers "Statistical Computer Performance Evaluation," edited by Walter Freiberger, Academic Press, 1972. (For a review of this book, see the ACM SIGMETRICS Performance Evaluation Review, V. 2, No. 1, March 1973, Pg. 16.)

In addition, books are reported to be in preparation by the following authors:

H. Hellerman, SUNY, Binghampton

P. F. Roth and M. Morris, Federal ADP Simulation Center

[1] University of California at Berkeley, Brigham Young University, and University of Washington.

[2] Now published, ACM-SIGMETRICS, Performance Evaluation Review, June 1973, Vol. 2, pages 37 thru 49.

3. Recommendations and Discussion

General—It is the author's belief that instruction in measurements should initially be concentrated through specific courses designed for that purpose, but that it should ultimately merge into other topics. Why concentrated at first; why not just let it grow as part of the other topics? For two reasons:

1. Such concentration will help to develop the principles and stimulate research activity in the area.
2. There is a need to train a "generation" of teachers and practitioners who are very conscious of computer measurements and evaluation so that they can assimilate these views into other activities.

Ultimately, however, courses on compilers, programming techniques, operating systems, computer networks, computer architecture, logical design, simulation and data structures should be taught with performance as an important underlying concept and measurements necessary to assess performance should be discussed as a normal part of the specialized topic.

As a side note, this relates to the author's belief that hardware and software monitors should flourish for a time, but they ultimately should be largely absorbed into hardware and software system designs. Measurement tools should be built into the systems right from the start. (If it can be done in Volkswagons it should be possible in computers.) Pressure from those who purchase and lease computers can bring this about.

Prerequisites—The current variation of computer science curriculum makes it inappropriate to make a hard and fast recommendation that measurements and evaluation should be at a graduate or undergraduate level. There is a general trend in computer science in which new curricular activities are initiated at graduate level but, as they become more well understood and as the depth and variety of undergraduate computer science background increases, the graduate topics flow down into the undergraduate area. At this time it is most likely that specific courses in measurements and evaluation would be initiated at the graduate level in most universities. More fundamental, however, is the question of prerequisites.

Is it better to have students well versed in the areas in which measurement and evaluation plays an important role before they take a formal course in measurements and evaluation? Or is it better to have them

aware of the importance of evaluation prior to studying the other areas? In view of the author's belief that measurement and evaluation viewpoints and techniques should ultimately be incorporated into other courses, the second approach seems preferable. However, it requires some preliminary appreciation of system architecture, operating system function and software systems structures before one can talk sensibly about measurements and evaluation. It seems appropriate at this time to concentrate on measurements and evaluation at an intermediate level after the student has gained an understanding of hardware and softwear system functions but before delving too deeply into design methods. Advanced work in measurements can still be added through thesis projects and individual study for the specialized student.

Some background in statistics should be acquired before focusing on measurement and evaluation techniques. Complexities of current systems often prevent use of direct causal relationships such as underlie many studies in the physical sciences, and one is forced to use some of the statistical tools long used by those in the social, physiological and medical fields. Students should have enough statistical background so that they can go on to understand what statistical concepts are useful in measurements and evaluation, why they are useful and why some common statistical techniques are not useful. (For example, some of the common variance reduction techniques become very costly when applied to data on computer performance.)

Measurements and evaluation of computers must be taught in conjunction with some kind of model of the systems being evaluated. A model is essential in order to provide the stimulus for measurement and evaluation questions and to provide a context within which results are used. As an absolute minimum, a "model" must exist in the form of someone's idea of how a system does or should perform. More formal modeling methods exist and their evolution and use in relation to computer evaluation should be encouraged.

The precise type of modeling that is emphasized in conjunction with measurements and evaluation is probably less important than that some modeling concept accompany the work. The emphasis on particular modeling techniques may vary from one university to another depending on the interest of the faculty. Some may concentrate on analytical models; others on stochastic or on simulation models. In all of these predictive modeling methods, measurements are necessary for establishment of parameter values and for validation of the models. Performance evaluation concepts are important in each of them in terms of orientation of the model and its use. Measurements and evaluation may also be related to purely descriptive models, which in general must precede the predictive models. In any case, the understanding of the system provided by descriptive models can serve as basis for planning of measurements and for application of the evaluative techniques.

Concepts—The following is a list of concepts suggested for inclusion in a course on measurements and evaluation:

1. Why measure?
   - 1.1 Types of measurements: Views of computer manager, user and designer.
   - 1.2 Results and how to use them: Tuning, equipment postponement and equipment selection.
2. Relation of measurement and evaluation to models.
3. Model taxonomies.
   - 3.1 Roles of various types of models in measurement and evaluation.
4. What to measure?
   - 4.1 Problem definition, its importance and its relation to measurements and evaluation.
   - 4.2 Know the system being measured. (Importance of this point should be stressed for all. The time in the course devoted to discussion of a particular system will depend upon the students' background and on the measurement project assignments to be given.)
   - 4.3 Influence of component cost on choice of measures.
5. Hierarchical approach to measurements.
   - 5.1 Focus on most important issues.
   - 5.2 Exogenous and endogenous systems.
6. Instrumentation (history of development, current status and future trends).
   - 6.1 Hardware monitors.
     - 6.1.1. Techniques of use: Probe attachment: setup and checkout.
   - 6.2 Software monitors.
     - 6.2.1. Accounting data.
   - 6.3 Advantages and disadvantages of the two monitor types.
   - 6.4 Combined hardware and software monitors.
7. Some underlying problems in time measurement.
   - 7.1 Implications of multiprogramming and multiprocessing.

7.2 Internal clocks: Tradeoff between resource use and timing accuracy.
7.3 Synchronization of external clocks.
8. Use of statistical methods for data reduction and experiment planning.
9. Measurement of system resource utilization.
9.1 System models (descriptive).
9.2 System profiles.
9.3 Throughput.
9.4 Use of hardware and software monitors.
10. Measurement of user programs.
10.1 Graph models.
10.2 Branch activity counts.
10.3 Branch execution times.
10.4 Time variation of memory requirement.
10.5 Turnaround.
11. Measurement of operating systems.
11.1 Relation to resource utilization and user program performance.
11.2 Operating system overhead.
12. Load description (for evaluation) and modeling (for prediction).
12.1 Need for load description to relate performance measurements to known conditions.
12.2 Methods for load description.
12.2.1 Traces.
12.2.2 Benchmarks and synthetic jobs.
12.2.3 Statistical measures.
12.3 Shortcomings of load description methods.
12.3.1. Lack of machine independence.
12.3.2. Costs of benchmarks and synthetic jobs.
12.4 Measures of computing work inherent in a program.
12.4.1 Status of research.
13. Future design for measurement.
13.1 Design features that could simplify the measurement tasks.
13.1.1 Hardware systems.
13.1.2 Software systems.

With only one text available, as noted above, (and it has not yet been reviewed relative to the recommendations of this paper) there is need for lectures in which the main objective is to provide perspective. Use of the current literature is important, but this must be done on a very selective basis since the volume of the literature has abruptly expanded since 1970. Direct project experience is extremely desirable, providing one has the appropriate vehicles, i.e. a computer with which to experiment and the hardware and/or software monitor. In some instances where it is impossible to experiment directly with the computer due to potential interruption of service to a large community of users, some experience can be provided by extracting information from the accounting log. This is a valuable but limited tool. Another important proviso concerning project experience is to select a task that provides insight but does not require an amount of work that proves overwhelming during a quarter or semester course.

Following are examples of some of the problems and projects assigned during various courses:

1. Student's FORTRAN program to extract and process a tape containing data on UNIVAC 1108 performance. The following data are plotted vs. elapsed time:
   Number of jobs in mix
   Memory use
   CPU time
   I/O request and time spent in I/O
2. Two-man projects are assigned on individual, real, system analysis projects drawn from local university, government or business administration applications (e.g. speed-up of access to personnel data; processing of parking tickets). No implementation is done; the final output is a report proposing a solution, emphasizing cost benefits. This provides good experience in planning, but lacks "feedback" from implementation.
3. Students inject assembly language counting and timing statements into FORTRAN programs, which are then run on an XDS Sigma 5. Resulting data are used to assess frequencies and times for program branches.
4. Students make use of pre-modified operating systems in Sigma 5 to intercept and analyze I/O calls.
5. Students compare hardware monitor measures of resource use in CDC 6400 with system accounting log measures of the same quantities—the simplest example being CPU utilization.

4. Summary of Recommendations

1. Measurement and evaluation viewpoints and techniques should be taught at the university level to spread consciousness of the importance of the

topics, and to encourage research and development.

2. Initially this should be taught as a separate topic, through formal courses and individual studies and projects. The ultimate aim should be toward inclusion in other courses where a measurement and evaluation view is important, and the separate course work should be needed only for advanced topics for students who specialize.
3. When taught as a separate course, measurement and evaluation should be placed at an intermediate level—after students are well aware of the function of hardware and software systems, but before advanced courses on design of such systems.
4. Familiarity with statistical methods should be acquired through prerequisites and should only have to be reviewed as part of the measurement and evaluation course work.
5. Measurement and evaluation should be taught in conjunction with course work on modeling of computer systems.
6. The particular type of modeling emphasized is less important than the viewpoint relating the modeling method to measurements and evaluation. The models (be they predictive or descriptive) should be used for their ability to provide evaluative information, and for their provision of a context for communications about measurements and their meaning.
7. Information should be exchanged on a continuing basis concerning the concepts to be taught and the most effective methods of conveying them. The SIGCSE Bulletin of the ACM provides one good forum for such exchange—probably better than the Performance Evaluation Review which is received by ACM SIGMETRICS members, who are already convinced of the topic's importance.
8. The most important recommendation is for continued research attention to develop the principles pertinent to measurement and evaluation of computers. This is a joint responsibility of those in industry, government and the universities.

This brief paper has gained much from the information and enthusiasm provided by the individuals contacted in other universities. It is hoped that this summary will, in turn, be useful to them and to others involved in this field.

Much of the author's opportunity to think about measuring and modeling techniques has been provided by support under grant #GJ28781 from the Office of Computing Activities, National Science Foundation, and this support is gratefully acknowledged.

3. Workshop Discussion

Due to time constraints, the education recommendations were not discussed as extensively as were the others. The discussion tended to underscore the importance of a strong educational program in CPE, as indicated by the following summary comment:

Browne: I would like to take the opportunity to raise a most significant point again. It was a general consensus that application of current technology for performance measurement and analysis could yield most of what improvement can be obtained. It, therefore, follows if by education we can motivate people to attempt performance evaluation while simultaneously training competent talent to carry out the tasks, a great impact could be made in only a couple of years. The figures collected at the conference would suggest a potential return of perhaps 1 to 2 billion dollars from such a program. Clearly then it should be of a very high priority.

APPENDIX A: LIST OF WORKSHOP PARTICIPANTS

NBS/ACM Workshop on Computer System Performance Evaluation

San Diego, Calif., March 27–30, 1973

Dr. Thomas E. Bell (Program Co-Chairman)
Software Research and Technology
TRW Systems Group
Mail Stop E1/1006
One Space Park
Redondo Beach, Calif. 90278

Dr. Barry W. Boehm (Workshop Chairman)
Director, Software Researrh and Technology
TRW Systems Group
Mail Stop E1/5017
One Space Park
Redondo Beach, Calif. 90278

Dr. James C. Browne
Department of Computer Science
University of Texas
200 West 21st Street
Austin, Tex. 78712

Mr. James H. Burrows
Headquarters, U. S. Air Force
Directorate of Data Automation
Washington, D.C. 20330

Professor Gary Carlson
Director, Computer Sciences
167 MSCB
Brigham Young University
Provo, Utah 84601

Mr. Walter Carlson
IBM Corporation
Old Orchard Road
Armonk, N.Y. 10504

Dr. Dennis Chastain
U.S. General Accounting Office
441 G Street, N.W., Rm 6108
Washington, D.C. 20548

Mr. George DiNardo
Vice President, Mellon National Bank, N.A.
Mellon Square
Pittsburgh, Pa. 15230

Mr. Gerald W. Findley
Automated Data & Telecommunications Service
General Services Administration
1121 Vermont Ave. N.W.,
Washington, D.C. 20405

Dr. Richard W. Hamming
Head, Computer Science Research Department
Bell Telephone Laboratories
600 Mountain Avenue
Murray Hill, N.J. 07974

Dr. Leo Hellerman
Dept. B10, Building 707
IBM Corporation
Poughkeepsie, N.Y. 12602

Mr. James Hughes
Xerox Corporation
701 South Aviation Boulevard
MIS A1–85
El Segundo, Calif. 90245

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

Dr. Robert R. Johnson
Vice President, Engineering
Burroughs Corporation
Second Avenue at Burroughs
Detroit, Mich. 48232

Dr. Stephen R. Kimbleton
University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina Del Rey, Calif. 90291

Mr. Philip J. Kiviat
Dept. of the Air Force
Federal Computer Performance Evaluation
and Simulation Center
Washington, D.C. 20330

Mr. Kenneth W. Kolence (Program Co-Chairman)
3591 Louis Road
Palo Alto, Calif. 94303

Mr. Robert L. Morrison
System Development Division
IBM Corporation
Dept. 99, Building 707–1
Poughkeepsie, N.Y. 12602

Professor Richard R. Muntz
Computer Science Division
School of Engineering and Applied Science
University of California, Los Angeles
Los Angeles, Calif. 90024

Dr. Norman R. Nielsen
Information Systems Group (J1053)
Stanford Research Institute
333 Ravenswood Ave.
Menlo Park, Calif. 94025

Professor Jerre D. Noe
Chairman, Computer Science Group
University of Washington
228 Roberts
Seattle, Wash. 98105

Dr. John Pasta
National Science Foundation
1800 G Street N.W.
Washington, D.C. 20550

Mr. Ray Rubey
Logicon, Inc.
Claypool Bldg., Suite 145
4130 Linden
Dayton, Ohio 45432

Professor Herbert Schwetman
Department of Computer Sciences
Purdue University
Lafayette, Ind. 47906

Mr. Eugene Seals
The Rand Corporation
1700 Main Street
Santa Monica, Calif. 90406

Mr. Dudley Warner,
Vice President & Chief Scientist
Tesdata Systems Corporation
1234 Elko Drive
Sunnyvale, Calif. 94086

Dr. Wayne Wilner
Burroughs Corporation
Hollister Avenue
Goleta, Calif. 93017

☆ U.S. GOVERNMENT PRINTING OFFICE: 1975 O—570-868

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBS-SP 406	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE

Computer Performance Evaluation: Report of the 1973 NBS/ACM Workshop

5. Publication Date: September 1975

6. Performing Organization Code: 640.00

7. AUTHOR(S) *S.Jeffery, Barry W. Boehm, Thomas E. Bell, Editors*

8. Performing Organ. Report No.

9. PERFORMING ORGANIZATION NAME AND ADDRESS

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Complete Address (Street, City, State, ZIP)

Same

13. Type of Report & Period Covered: Final

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 75-619080

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.*)

An ACM/NBS Workshop on Computer Performance Evaluation (CPE) was held in San Diego, California in March, 1973. The papers, workshop discussions, conclusions and recommendations presented in this volume address specific problems in making computer performance evaluation a commonplace and productive practice.

While several of the conclusions indicate that improvements are needed in performance analysis tools, another suggests that improved application of CPE could be achieved by better documentation of analysis approaches. More integration of data collection and modeling are considered necessary for the performance analysis field to develop its full potential. Participants noted that the common emphasis on data collection or modeling, to the exclusion of considering objectives, often seriously degrades the value of performance analysis.

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 architecture; computer performance evaluation; installation management; measurement; modeling; monitoring; operating systems; performance objectives.

18. AVAILABILITY

[XX] Unlimited

[ ] For Official Distribution. Do Not Release to NTIS

[XX] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13.10:406

[ ] Order From National Technical Information Service (NTIS) Springfield, Virginia 22151

19. SECURITY CLASS (THIS REPORT): UNCLASSIFIED

21. NO. OF PAGES: 175

20. SECURITY CLASS (THIS PAGE): UNCLASSIFIED

22. Price

USCOMM-DC 4...

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH reports National Bureau of Standards research and development in physics, mathematics, and chemistry. It is published in two sections, available separately:

• Physics and Chemistry (Section A)

Papers of interest primarily to scientists working in these fields. This section covers a broad range of physical and chemical research, with major emphasis on standards of physical measurement, fundamental constants, and properties of matter. Issued six times a year. Annual subscription: Domestic, $17.00; Foreign, $21.25.

• Mathematical Sciences (Section B)

Studies and compilations designed mainly for the mathematician and theoretical physicist. Topics in mathematical statistics, theory of experiment design, numerical analysis, theoretical physics and chemistry, logical design and programming of computers and computer systems. Short numerical tables. Issued quarterly. Annual subscription: Domestic, $9.00; Foreign, $11.25.

DIMENSIONS/NBS (formerly Technical News Bulletin)—This monthly magazine is published to inform scientists, engineers, businessmen, industry, teachers, students, and consumers of the latest advances in science and technology, with primary emphasis on the work at NBS. The magazine highlights and reviews such issues as energy research, fire protection, building technology, metric conversion, pollution abatement, health and safety, and consumer product performance. In addition, it reports the results of Bureau programs in measurement standards and techniques, properties of matter and materials, engineering standards and services, instrumentation, and automatic data processing.

Annual subscription: Domestic, $9.45; Foreign, $11.85.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a world-wide program coordinated by NBS. Program under authority of National Standard Data Act (Public Law 90-396).

NOTE: At present the principal publication outlet for these data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St. N. W., Wash. D. C. 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The purpose of the standards is to establish nationally recognized requirements for products, and to provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Federal Information Processing Standards Publications (FIPS PUBS)—Publications in this series collectively constitute the Federal Information Processing Standards Register. Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service (Springfield, Va. 22161) in paper copy or microfiche form.

Order NBS publications (except NBSIR's and Bibliographic Subscription Services) from: Superintendent of Documents, Government Printing Office, Washington, D.C. 20402.

BIBLIOGRAPHIC SUBSCRIPTION SERVICES

The following current-awareness and literature-survey bibliographies are issued periodically by the Bureau:

Cryogenic Data Center Current Awareness Service

A literature survey issued biweekly. Annual subscription: Domestic, $20.00; foreign, $25.00.

Liquefied Natural Gas. A literature survey issued quarterly. Annual subscription: $20.00.

Superconducting Devices and Materials. A literature survey issued quarterly. Annual subscription: $20.00. Send subscription orders and remittances for the preceding bibliographic services to National Technical Information Service, Springfield, Va. 22161.

Electromagnetic Metrology Current Awareness Service Issued monthly. Annual subscription: $100.00 (Special rates for multi-subscriptions). Send subscription order and remittance to Electromagnetics Division, National Bureau of Standards, Boulder, Colo. 80302.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington, D.C. 20234

OFFICIAL BUSINESS

Penalty for Private Use, $300

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-215

SPECIAL FOURTH-CLASS RATE
BOOK