N71-24023

NASA CR-118037

R-584-NASA/PR

February 1971

COMPUTER PERFORMANCE ANALYSIS: MEASUREMENT OBJECTIVES AND TOOLS

T. E. Bell

A Report prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
AND
UNITED STATES AIR FORCE PROJECT RAND

Rand
SANTA MONICA, CA. 90406

This research is sponsored by the National Aeronautics and Space Administration under Contract No. NAS-12-21-44 and the United States Air Force under Project Rand— Contract No. F44620-67-C-0045— monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of Rand, NASA or of the United States Air Force.

R-584-NASA/PR

February 1971

COMPUTER PERFORMANCE ANALYSIS: MEASUREMENT OBJECTIVES AND TOOLS

T. E. Bell

A Report prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
AND
UNITED STATES AIR FORCE PROJECT RAND

Rand
SANTA MONICA, CA. 90406

Published by The Rand Corporation

PREFACE

This report represents one in a series of studies on computer performance analysis. As computers have increased in cost and complexity, the Air Force and NASA have increased their efforts in performance analysis. Analysts in each new increment of effort must decide whether formal performance analysis is justified and, if so, what tools to use in the study. This report's purpose is to aid present or potential analysts make such decisions so that they can proceed to a number of other critical topics in performance analysis to maintain high cost-effectiveness of computer systems.

Although initial work for this report was sponsored by NASA, the Air Force's large investment in computers and increasing interest in performance analysis led to continuation of the work under Project Rand.

SUMMARY

This report suggests a number of objectives for computer system measurement and analysis beyond the commonly accepted one of tuning. Objectives in computer operations--identifying operational problems and improving operational control--mean that personnel in this area should become familiar with the new tools and techniques. Computer system simulators should be concerned with model validation as well as model development. Installation managers need results from this field in order to select equipment, trade man-time for machine-time, and tune installed equipment.

Data collection tools for use in measurement and analysis are necessary to fulfill these objectives. These tools range from simple, inexpensive ones--audio and visual indicators, operator opinions, and logs--to the more sophisticated hardware and software monitors. Each of the simple tools can provide initial indications of performance, but hardware and software monitors are usually necessary for a thorough analysis. Five binary characteristics can describe a monitor: (1) implementation medium, (2) separability, (3) sample portion, (4) analysis concurrency, and (5) data presentation. An analyst should determine the characteristics his analysis requires before choosing a product.

Recognizing objectives and choosing measurement tools are two important steps in a performance analysis study. This report deals with these two topics so that analysts can proceed to four more difficult and critical topics. Modeling, choosing a data collection mode, experimental design, and data analysis deserve at least as much attention as examining data collection tools.

CONTENTS

I. INTRODUCTION

The necessity for some type of performance evaluation in computer systems is becoming widely recognized. Some recently announced computer systems include hardware monitors to aid in determining performance. The measurement portion of the American computer industry is growing rapidly; even the relatively less-advanced Russian AIST-0 operating system[†] will include integral performance measurement.

Work in performance evaluation tends to emphasize the development and choice of measurement tools that improve performance. Unfortunately, this emphasis diverts effort from other important issues in performance analysis and concentrates interest almost exclusively on improving the performance of operational systems.

Typically, an installation that undertakes performance evaluation begins by reviewing the available measurement tools. Each vendor's product is evaluated and one product is finally chosen. The tool is then used to gather data, which often are not in a useful form. Some criteria are needed so that an appropriate choice of tools can be made and effort concentrated on planning data collection and analysis. Section II discusses the various reasons for considering performance analysis. Determining the appropriate objectives should lead to more effective use of tools as well as to better analysis.

[†]A time-sharing system being developed in Novosibirsk at an Institute of the Soviet Academy of Sciences.

II. WHY MEASURE AND ANALYZE?

Measurement and analysis of a computer system are usually expensive. Data collection hardware and software must be developed or purchased, the system must be disrupted during tests, and even monitoring during normal operations often slows processing. Personnel must plan and execute experiments, reduce data (and correct erroneous records), agonize over results as conclusions are formed, and implement indicated actions--the rationale for the entire process.

In addition to being expensive, this process disrupts the normal interaction between operators and computer. Nevertheless, interest in measurement and evaluation is increasing. The advent of third-generation systems marks the end of the period when human abilities were adequate to understand operations without formal measurement and analysis. Today, systematized procedures are necessary for good operational control, equipment selection, and system tuning.[†]

IDENTIFYING OPERATIONAL PROBLEMS

In second-generation batch systems, identifying operational problems involved examining elapsed machine times. With third-generation multiprogramming, multiprocessing, real-time, on-line, or time-shared systems, this simple analysis process is inadequate. The volume, speed, and concurrency of operations are too great for elapsed times to have more than superficial meaning. In the past, an entire system handled only one job at a time; today, resource sharing is a normal operational mode and resource denial is on a probabilistic basis.

[†]Tuning is the process of measuring the system, understanding effects, and making small changes in hardware and software that cause large increases in system performance.

Operators or systems programmers can easily (and even accidentally) alter a system so that large parts are unavailable. For example, changing a disk's designation can disallow its use for normal operation; the changed designation may remain undetected for long periods. Job characteristics continually change; jobs run differently in different combinations with other jobs. Small variations in operator response can cause large variations in system response. Resource allocation in a large system does not depend on job characteristics alone; this complex dependency causes variability in run characteristics of identical jobs.

In determining minimum variability in Rand's IBM System 360/65, we used Release 17 of IBM's OS/MVT to run a number of identical jobs under a variety of conditions. The standard Rand accounting system determined run characteristics of program execution. The program performed I/O to and from system-allocated disk space and also used the timer to cause frequent interrupts. When run in an otherwise idle system, each job required an average of 15.11 sec of CPU time and performed 7765 I/O operations. The standard deviations of these values were 3.2 percent and 1.7 percent of the averages; the ranges were 8.6 percent and 7.8 percent of the averages. The only sources of variability were resource allocation and low-level timing relationships. When other sources of variability are active, changes several times this size are observable. The noise (inherent random variability) in a system's operation is very large when only gross measures are used. Analysts often need to use special, specific metrics that have lower noise levels so that important trends can be detected. These special measures can then be used to differentiate between human operator failings, inadequacies in the operating system, and inappropriate loads from users.

OPERATIONAL CONTROL

Management can often schedule loads more efficiently if it can determine the system-wide effects of certain job loads.

Instead of concurrently running similar jobs, heavy I/O jobs can be concurrently run with heavy CPU jobs. If total demands on I/O, CPU, core, etc., are constant, machine capacity increases with mixed resource loading. However, this is feasible only when detailed job characteristics are known and understood.

Good accounting also requires an understanding of loading effects. Charging should be at least partially based on system usage, e.g., on denial of the system to others. Significant effort is often required to determine the system-wide effects of a particular load on a particular system. After these effects become known, management may choose to alter charging and scheduling methods to bias users away from a load that adversely affects the system. Of course, such a step should be taken only after measurement and analysis have proved its correctness.

SIMULATION DATA

People simulate computers to (1) predict performance of a new or altered configuration, (2) check their understanding of a system, (3) determine the effect of new software, or (4) obtain information on system capacity. A simulation is valuable in achieving these objectives only if it reflects reality, i.e., only if the input data accurately reflect the real world and the simulation's logic results in data similar to those obtainable from real systems.

Obtaining these data for use in simulations constitutes an important (and often neglected) application of measurement and analysis. Although massive efforts are often devoted to creating simulations, model validation is usually not considered worth the added 10- to 20-percent expenditure of funds needed for a preliminary validation exercise.

This emphasis is usually understandable; in some instances, it may even be justified. If a simulation aims at increasing general understanding of interactions, validation beyond code examination may not be cost-effective.[†] However, this is usually not the reason for the lack of interest in validation. Measurement and validation techniques are quite different from programming, and inexperience usually constitutes a psychological barrier that prevents the simulator from performing even perfunctory measurement. Instead of gathering his own data (the way he programs), the simulator proposes that someone else do this--a request usually met with distinct disinterest. More measurement competence among simulators could alleviate the present situation of simulation without validation.

DEFINING SIMULATION MODELS

Pre-simulation measurements can be at least as useful as post-simulation validation measurements. Even very detailed simulations are models--abstractions that explicitly recognize a few system characteristics, approximate some, and ignore the rest. The simulation's usefulness depends on consideration of appropriate characteristics; for example, even the most accurate representation of CPU operations will not be adequate to predict the throughput of an I/O-bound system.

We performed a very detailed simulation of Rand's Video Graphics System (VGS) that included the effects of a Special-Purpose Multiplexer on an IBM 1800.[‡] We began by examining the specifications of the Special-Purpose Multiplexer, which led to a very simple model consisting of a single transmission rate. Because unquestioned acceptance of such specifications can be risky, we decided to measure actual performance.

[†]Simulation can be a valuable tool for checking overall understanding of a software system. The discipline of simulating the system can reveal many misunderstandings. If the objective is to uncover these problems, a check of the system's code is often all the validation necessary.

[‡]See Ref. 1 for a description of this effort.

The 1800's Special-Purpose Multiplexer communicates with an IBM 360/50 through an IBM 2701 to pass messages between the two computers. We set the 360 to transmit for 60 sec and measured the amount of data transfer. Then, we set the 1800 to transmit for 60 sec and measured data transfer. We repeated the experiment under several conditions; it strongly indicated that the data-transfer rate was not a simple, constant value. Instead, due to the particular hardware configuration, the data-transfer rate depended in a complex manner on the 360 interface unit. This experiment enabled us to identify the important variables and use a more appropriate model.

We also analyzed data from our IBM 360/65 and determined that a scheduling simulation of this multiprogrammed machine was inappropriate. Our simulation implicitly assumed that main storage was the most important system resource, i.e., that having more jobs in core results in more throughput. Data from the real system convinced us that the number of jobs in the machine during normal operations did not significantly affect throughput. Because our machine was very strongly I/O-bound, a simulation that concentrated on I/O was far more appropriate than one that considered core storage.

The effort involved in pre-simulation data analysis should *not* be considered useful just for orienting simulation work. The simulation should provide the same sorts of data as the real system, at least for validation. Therefore, data-analysis techniques developed for the real system are applicable to the simulation data. The reverse is also true: analysis tools developed for simulation data are often applicable to data from the real system. Combining the techniques developed in these two areas leads to improved data analysis in both.

Building simulations around measured and interpreted data increases the utility of the simulation more than enough to justify the measurement effort.

EQUIPMENT SELECTION

Equipment selection and acquisition are important management responsibilities that cannot be delegated to an outside organization, particularly a vendor. Vendor representatives have an inherent bias toward solving problems by equipment changes (usually equipment upgrades). However, changes in the operating system or in allowed user loads may solve some problems more efficiently. Measurement and analysis by installation personnel are imperative for rational equipment selection.

Selecting an entirely new system is more critical and challenging than incrementally analyzing an existing one. A new system will have a new set of capabilities and limitations; loads shift and many assumptions become invalid. Therefore, complete measurement and analysis of both the old and the new system are particularly necessary in order to predict a new system's operation. A gross measure, e.g., the time to process a given set of jobs, can be extremely deceptive; vendors might carefully arrange the job stream to minimize elapsed time on the proposed system. Other indicators can determine whether this is the case.

Some important questions are:

- Will a new compiler sufficiently change system loading so that new equipment acquisition can be avoided?
- Are users unknowingly generating loads that have inappropriate characteristics for the present configuration?
- What are the current bottlenecks in system operations? (If only one resource is limiting, only that one should be purchased.)
- If new equipment must be acquired, how many units should be obtained? (Excessive acquisition of one type of unit leads to a shifting of the bottleneck and under-utilization of newly obtained resources.)

- o Can the present configuration be downgraded without impacting capacity? (Excess resources may be returned without significant effect on the system.)
- o Are quoted CPU, channel, and device utilizations for a proposed system within the usual range of this manufacturer's systems in the field?
- o How would a new system react to loads that differ from present ones?

TRADING MAN-TIME FOR MACHINE-TIME

A number of techniques are available that allegedly increase the efficiency of personnel using the computer but that decrease the machine's efficiency. Time-sharing, text editors, interactive graphics, and conversational programming systems are loved by the users and hated by the operators because they aid the former but reduce the machine's capacity.

Providing user aids at the expense of machine efficiency is one extreme; an opposite philosophy forces users to act in ways that improve machine efficiency. Universal use of one language, large blocking of test output, and carefully designed code increase machine efficiency but cost users an abnormally high effort to achieve any results.

In general, there is a tradeoff between the users' time and the computer's efficiency. Only factual analysis of system effects on both human and machine performance will resolve the tradeoff problem. This report deals only with measuring machine performance.

TUNING

Third-generation computer systems tend to be very sensitive to minor changes.[†] One of the most typical problems

[†]Strangely, this topic is considered controversial in many computer installations. Systems programmers often take the fatalistic attitude that the systems are no good and that improvements are infeasible. The literature, however, indicates otherwise. Bemer and Ellison refer to a

in multiprogramming is I/O contention, where several jobs have concurrent requests to the same I/O devices. The I/O devices (usually disk) must then respond to each in sequence (usually by performing seeks). The total seek time greatly exceeds total data-transfer time. Running alone, a single job positions the disk head only a few times and transfers data at a higher overall rate.

In time-sharing systems, the time spent setting up jobs (swapping) and deciding what to do next (performing task switching, determining time quantums, etc.) can easily exceed the time spent on useful computation. Certain policies are particularly dangerous in this environment because they tend to aggravate bad situations. For example, time-sharing efficiency can be very sensitive to the amount of high-speed memory allocated to each user. As the size decreases, thrashing may begin and processor utilization decreases. One response is adding more jobs in an attempt to use the excess CPU. This policy further reduces the memory for each job and aggravates the inefficiency.†

Mitigating such third-generation problems requires both understanding potential problems and performing specific measurements to indicate which problems are actually occurring. A basic characteristic of computer systems is that demands change over time. Because of shifting loads, tuning (including measurement and analysis) must be a continuing

---

". . . 4 to 1 leverage on system productivity" [2, p. 39]; Johnston notes an increase of 500 percent in CPU utilization of a 360/91 [3]; Saltzer and Gintell state that ". . . the payoff in being able to look at the meters any time a performance problem is suspect . . . is very high" [4, p. 500]; Wiener and DeMarco ". . . have come to the conclusion that most medium-sized systems with general-purpose work loads, run at less than 30 percent CPU efficiency even when they are running" [5]. They also note that the best systems perform at 75- to 80-percent efficiency.

†For an explanation of this problem, see Ref. 6.

process.[†] Intensive effort must be occasionally invested to revise operating assumptions that may have become decreasingly true. In addition, after each shift in hardware and software configuration, intensive effort must be devoted to tuning. This includes measurement immediately after an attempt to improve system performance. All too often, a change has related, unanticipated effects elsewhere in the system.[‡]

[†]Bemer and Ellison state that "instrumentation should be applied to software with the same frequency and unconscious habit with which oscilloscopes are used by the hardware engineer" [2, p. 39].

[‡]For example, rounding up segment sizes to an integral number of page sizes may reduce external core fragmentation but cause less efficient use of core [7].

III. DATA COLLECTION TECHNIQUES

Simple indicators of system performance furnish much less information than more sophisticated ones, but also cost much less. Understanding and using both kinds of indicators should be considered important.

AUDIO AND VISUAL CLUES

Each time a piece of unit record equipment processes a record, it emits a sound. The systems analyst can use this sound to roughly estimate activity and judge the occurrence of certain system-wide problems. For example, a multiprogrammed system may be experiencing severe disk contention in attempting to print spooled records.[†] Quite often, this problem manifests itself in strongly synchronized printing from the several printers on a large system. As the disk head moves from track to track, first one then another printer operates. When one printer completes output for its job, the other printer(s) begins operating at a sharply increased rate.

Usually, the manufacturers provide visual clues in the form of incandescent or neon lights. "Watching the lights blink" can be a useful pastime for the analyst. On IBM

[†]"Spooling" involves the temporary storage (usually on disk) of input or output for a job during job execution. Input is stored prior to initiating execution and output is put in its final form (usually printed or punched) after execution. In multiprogrammed systems, this keeps input and output for each job in known places. Since several jobs are simultaneously proceeding to completion, outputting to a printer would cause printing for one job to be interspersed with printing for all others; an overwhelming cutting and pasting job would result. Instead, a job's total output is put in a special file on disk or tape; other jobs have their output placed in other files. After a job has completed execution, its total output is printed (and/or punched), at one time, by special utilities. In addition, spooling allows faster devices than printers or punches to be used for outputting while that job is in core. The slower, final output of data can then be done by a small, efficient utility routine.

System 360s, WAIT and SYSTEM lights are usually the most sharply observed. The CPU is idle if the WAIT light is on; some part of the system is active if the SYSTEM light is on. Because the lights are incandescent, they crudely integrate system activity over time. A very bright WAIT light with a blinking SYSTEM light usually indicates an inefficiently used CPU--an I/O-bound system. A system operating in multiprogramming mode with a solid SYSTEM light and a very dim or completely dark WAIT light usually means a CPU-bound system, one making very good use of its CPU (or hung in a coding loop). Other lights on channels and controllers indicate activity on these resources.

The availability and obviousness of lights are too good to be true. We carefully measured the actual activity of various interrupt levels on an IBM 1800 and also watched the console lights. We were barely able to visually detect the difference between one light, active 0.3 percent of the time, and the one next to it, active 1 percent of the time. We were able to detect when one light was switched from 35-percent activity to 70 percent, but were unable to visually determine when activity increased to 100 percent. Visually perceived light intensity is highly non-linear with actual activity; great caution should be used in interpreting the lights.

Multiple, rapidly spinning tapes and extremely active disk heads can, in some environments, indicate severe trouble. In other environments (where loads should be causing this kind of behavior), they may indicate a smoothly running system. Unfortunately, most installations fall somewhere between these two extremes, leaving managers with an amorphous feeling of unease. The simple clues are valuable first impression devices; however, more substantive data are usually needed to meet analysis objectives.

OPINIONS OF OPERATING PERSONNEL

Because operators are in intimate contact with the machine, they are usually one of the primary sources of data. For example, attempts to improve physical layout and decisions on where to place unit record equipment, tapes, and consoles can greatly benefit from their opinions.

Unfortunately, operators are not unbiased observers. They are responsible for the system's continued operation and they unconsciously weight trouble-causing jobs more heavily than jobs that require no special operator activity.[†] Good operators make sure the job flow is steady and, within available alternatives, they have the jobs processed as quickly as possible.

At Rand's Computation Center, operators use the number of jobs completed as the primary indicator of system performance. Typical cumulative numbers for each hour of the day are compared with the actual number to determine performance. Obviously, this indicator is very sensitive to job mix, hardware failure, and simple randomness in sequencing, but it has the distinct advantage of availability.

LOGS

Most installations have printed forms on which users enter job names, submission time, and a few job parameters. These forms also commonly record the exit of the job from the computer center by adding the time of sign-out (making output available to the user). These logs can be used to determine total turnaround time in the computing center (as

[†]Peter Blau, investigating the activities of office personnel faced with a multitude of objectives, found that a few objectives were picked by the personnel as important; the rest were ignored (see p. 231, note 9 of Ref. 8). Computer operators seem to act similarly.

opposed to turnaround time within the machine itself). The data are also used to determine how long users take to make corrections before resubmitting a job.[†] Although sign-in logs are considerably more reliable than an individual's subjective evaluation, they are often questionable due to poor handwriting and inaccurate data entry (users continue to misread wall clocks).

Console logs produced by the machine often contain very interesting information about the man-computer interface; however, this information is buried in such a quantity of data that manual extraction becomes infeasible. Although information for general analysis may be too difficult to access, the console log can be very valuable for detailed data about specific interactions.

Audio clues, visual clues, operator opinion, and logs can only provide gross performance indicators. On the other hand, hardware and software monitors can provide detailed data on internal system operations. However, an increase in the cost of the data collection device matches this increase in data.

HARDWARE MONITORS

A hardware monitor obtains signals from a host[‡] system through high-impedance probes attached directly to the computer's circuitry.[††] The signals can usually be passed through logic patchboards to do logical ANDs, ORs, and so on, enabling the analyst to obtain signals when certain

---

[†]This assumes that users give all jobs unique identifiers rather than using a few identifiers for all jobs. We found that batch users at The Rand Corporation do the former.

[‡]The host machine is the device being monitored.

[††]For some specific descriptions of commercially available units, see Refs. 9-13.

arbitrary, complex relationships exist. The signals are then fed to counters or timers. For example, an analyst with a hardware monitor could determine (1) the portion of CPU time spent performing supervisory functions while only one channel is active, or (2) the number of times a channel becomes active during a certain period.

Either the monitor or the host machine often contains facilities for doing simple comparisons. This capability is usually applied to address comparisons; the address of the currently executing instruction is compared with two bounds registers in the monitor to determine whether a particular set of code is being executed. Because the values in the bounds registers must be set manually, the location of the desired code must be known. One hardware monitor [14] includes sequence-testing circuitry to determine when certain sequences of instructions are executed. In some cases, these monitors have been designed to examine such specific operations as the incidence and type of jump instructions [15]. Normally, however, sequence-recording is beyond the monitor's capability.

Because hardware monitors can sense nearly any binary signal, they can be used with a variety of operating systems, even with machines built by different manufacturers. This characteristic can be important in an installation with a variety of machine configurations. Of course, personnel must be familiar with each system measured, but a new measurement tool is not needed for each case.

Hardware monitors are able to obtain accurate, high-resolution data without affecting the host system in any way. They inject no artifact[†] in the system and thus can

[†]"Artifact" is the perturbation in system performance due to the presence of the measuring device. On occasion, hardware monitors have been rumored to introduce an artifact by loading computer circuits and slowing response. Such instances are very difficult to verify.

be activated and deactivated without affecting processing rates. On many such monitors, a relatively small number of counter/timers are available (often 16) and only a small number of probes can be used (often 20).[†] As a result, only a few items can be checked simultaneously. Because the locations of items in the core of a third-generation machine are usually not known in advance, queue lengths cannot be determined unless some programmable, relatively sophisticated monitor is used. Similarly, it is usually not possible to measure a particular program's activity because the monitor does not know the association between an action and the activity-causing program.

Self-Monitoring

One manufacturer of specialized computers, Comcet, Inc., has integrated 256 probes into its hardware and software. The probes collect data on usage and maintain 32 visual displays for on-line use. These displays are up-dated 60 times/sec and the results recorded for possible subsequent analysis every 5.2 sec. Data collection points include wait state, worker state, interrupt state, storage activity, channel activity, line activity, and various optional software activities.

IBM has provided some hardware monitoring capability on its 360/85 and 370/165 computers. Probes are available to indicate CPU and channel activity, and certain logical combinations are possible by operator-settable switches. Output is primarily through a moderately damped meter, but any one probe can be reached through a hub for external monitoring.

[†]On some monitors, however, many more counter/timers and probes can be used.

Monitoring With Another Computer

There has been interest in monitoring a host computer with another computer (a very sophisticated hardware monitor). UCLA's SNUPER computer work [16] falls in this category but requires more extensive effort than usually available. R. W. Murdy of IBM has reported on an associative memory[†] device that monitors and concurrently reduces data on such machines as the IBM 360/40 [17]. However, this effort is presently in the category of a research tool. An alternative to the large SNUPER computer monitoring technique or associative memory tools is a mini-computer with attached probes and some standard software. On command from an attached console, such a monitor could collect and process specific data and output results.

Data Collection

Hardware monitors have their own clocks, which operate independently of the interval timer on the host machine. Timings on multiprocessors can become ambiguous because such systems usually contain two or more timers, one for each CPU. The independent clock on the hardware monitor then becomes a standard for measurement.[‡]

Since hardware monitors operate independently of host computers, system catastrophies do not terminate data collection. As a result, when other techniques fail, the causes of such unfortunate instances can be identified with hardware

[†]An associative memory has the capability of simultaneously comparing an input number with the contents of all memory locations and returning the location of any word with equal value. These devices tend to be very expensive.

[‡]See p. 496 of Ref. 4 for a discussion of the utility of independent timers. Since interval timers on many machines have very low resolution (16.7 msec typically), they are of limited value for such detail timings as the time to execute a certain piece of code.

monitors. Operator errors are particularly susceptible to detection by hardware monitoring.

Collected data are of three types: time, utilization, and counts. Measured times may include either the total time a certain condition exists or the elapsed time between occurrences.

Full-Time Monitoring

Most commercially available monitors record everything that happens within the limits of their resolution. These devices monitor during the full time they are connected; there are as many simultaneous paths as there are probes. If connected to signals with pulse repetition frequency greater than their maximum speed, these monitors behave in undetermined ways. This seldom represents a real problem since the maximum frequency is usually 1.0 to 1.3 MHz.

Sampling

Some hardware monitors sample, i.e., they only occasionally examine the trueness of logical conditions. A mini-computer monitor that uses this technique samples at a relatively high rate (about 100 times/sec); therefore, results over several minutes are highly representative of the results that would be obtained with full-time monitoring. Having sampling times determined independently of the monitored machine avoids synchronization with internal operations. The Comcet 60 (which samples functions at 80- or 640-μsec intervals with integrated hardware probes) solves the synchronization problem by sampling based on a second clock, independent of the clock used in the remainder of the system.

Data Reduction

Present monitors often write results of counters and timers on tape for subsequent processing. Writing to tape often overlaps data collection, so dead time is only a few microseconds instead of the unoverlapped time of 10 to 20 msec. Visual display is an alternative to tape output often used in slower, lower-cost units.

Usage

Because hardware monitors are rather expensive, most installations rent one for a few days. The real problem is knowing what to do before the hardware box arrives. The necessity for exhaustive test planning cannot be overemphasized; inadequate planning can result in a costly mess. One strategy is to experiment with a unit for a week, then plan to use it for another week after a month for planning has elapsed. During this month, the analyst attempts to understand results, improve exerciser programs that insure correct monitor functioning, and generate a very specific set of performance tests. This strategy causes personnel to be very busy for two weeks collecting data in the hope that data reduction will be reasonably well-defined. However, this procedure has some severe drawbacks since errors in measuring cannot be corrected after the monitor is gone.

SOFTWARE MONITORS

Software monitors are often able to do things hardware monitors cannot, but they also often have a large effect on system performance and fail to measure some items accurately. These monitors consist of code residing in the memory of the host machine (at least for short periods); in some cases, they may require as much as 20 percent of CPU capacity and 10 percent of I/O capacity. Because of this large effect, software monitors are carefully designed in an attempt to minimize their resource requirements.

Because software monitors reside in memory, they have access to all tables that the system maintains. Therefore, measurement of internal program operations is less difficult and the monitor can examine core usage, queue lengths, individual program operation, and so on. Conceptually, most states and processes that interest a programmer can be measured if he is willing to pay for them in artifact.

Nevertheless, the answer to all problems is not merely substituting a software monitor for a hardware monitor. Because software monitors reside in core, they must coexist with any other programs (e.g., control programs). This places certain limitations on a monitor's design. The operating system may mask[†] interrupts that the monitor cannot unmask without risking frequent system failures. Because interrupts may be masked, completion of an I/O operation may not be recorded until long after it occurs. In practice, this error is usually small, within one or two percent.

The precision of measurements can be no greater than that of the accessible timer in the host computer. This often means that the maximum precision of measurements is 1/60 sec. This may be sufficiently precise, but higher-resolution timers are often necessary and must be added to a machine.

Software in the CPU does not have access to much information about hardware activity at the other end of the channels. Timings on these activities are usually by inference and make use of certain approximations. Hardware monitoring must be used if very accurate measurements are needed.

[†]"Masking" in this case refers to a computer's ability to ignore interrupts, either permanently or for a short period. Various machines allow masking of specific types, classes, or hierarchies of interrupts. Operating systems often mask interrupts to perform sensitive operations; if a software monitor unmasks, the operations can be interrupted, which may cause system failure.

and data reduction can be partly done on-line. This technique appears similar to a sophisticated hardware monitor, but a significant difference exists: a hardware monitor examines a system in its normal state; a semi-separate, software monitor examines a system that has been degraded so that one peripheral processor is unavailable.

Tracing

Tracing involves recording every time a certain class of operations occurs; the events trigger the data collection. In some cases, this "represents a complete timed record of literally everything significant that . . . occurred."† In a 5-min run, this technique collected 350,000 trace entries on tape when applied to the GECOS II operating system for the GE 625/635. A far less extensive tracing is pursued by AMAP, an IBM proprietary tracing program used on IBM System 350 hardware when operating OS/MVT [20]. in AMAP, I/O is emphasized and each I/O operation recorded. Other tracing programs record far fewer data, with a consequent decrease in artifact.

Tracing programs use internal interval timers; the precision of the resulting data can be no greater than that of the timer. A great deal happens in a system between two consecutive timer increments; all the events are recorded as occurring at one time. Therefore, a high-resolution timer is valuable in generating traces.

Cantrell and Ellison note that their extensive trace program represents a one-percent load on the system [21]. However, this low overhead may not compensate for tracing's most severe drawback: the large space required for storing output. The huge quantity of data requires that only a relatively small period of operation be examined (typically,

†See p. 214 of Ref. 2.

Accounting Systems

The most common software monitors are accounting-type systems, which aggregate usage by task, job, or other unit of user work. The primary objective of the accounting system's designer is cost allocation, which sometimes compromises the usefulness of accounting system data, particularly where overhead is involved.

Accountancy may require that computer overhead costs be borne by users who are charged directly for their demands on the system. One way to do this without annoying users is to include much of system overhead in usage charges. (Users almost never know what their charges would be without overhead, and this overhead is necessary for job execution.) Unfortunately, the portion of total job time for overhead can be in excess of 50 percent and the degree of overhead is highly dependent on operations elsewhere in the system. One study found that, on certain jobs, accounting times could be in error by 220 percent due to allocations of overhead time [18].

Although accounting data can be deceptive, analysts can determine the actual charging methods used and perform analyses based on a good understanding of potential errors. Accounting data also have some distinct advantages. They are usually quite complete because they are retained for historical purposes, and changes in collection methods are well documented so that users can examine them for correctness.

Semi-Separate Monitor

On certain types of computers, a monitoring program can be put in a peripheral processing unit and have very little influence on the rest of the system.† A large number of accessible items can be examined whenever a change occurs,

†This has been used on CDC 6000 series equipment [19].

2 to 30 min) in order to ensure that the limits of the recording medium (usually tape) are not exceeded. The short recording period and potentially large impact on a system's operation impair the technique's potential usefulness.

Sampling

Many types of analysis do not require the very detailed data obtainable from tracing but do need reasonably detailed data obtained over a relatively long period. Accounting data do not have the necessary detail, and tracing cannot be efficiently performed over a long period. Sampling can often be used in this situation.[†]

Such fields as accounting and management science have long used random sampling for approximating the characteristics of a population. Techniques are readily available to predict the reliability of estimates made through unbiased random sampling plans.[‡] Such situations necessitate few assumptions about the system under examination; the randomness used in picking the sample counteracts the effects of cyclic behavior and gives each system state a chance of being examined in proportion to its frequency and duration.

In computer monitoring, sampling is performed by interrupting the CPU at regular intervals so that the sampling program can examine the system. Special software in the computer responds to interrupts by storing in registers or core all values that are important to the program in control at the time of the interrupt. Since this interrupt-handler software stores all important values in well-defined locations, the sampling program can examine these data. If the

[†]Commercially available sets of sampling tools for IBM System 360 equipment are available (see Refs. 12, 22, and p. 9 of 23 for some examples). Another set of sampling tools has been developed at the Stanford Linear Accelerator Center(SLAC) [3].

[‡]See, for example, pp. 212ff of Ref. 24 and pp. 744ff of Ref. 25.

CPU is idle when interrupted by the sampling program, the interrupt handler will have data stored indicating CPU idleness. The sampling program can usually determine which code was being executed prior to the interrupt. It can also determine the status of channels, controllers, and devices (and the cause for that status), as well as other information of interest to analysts.

In conventional industrial sampling, randomness (essentially, the inability to predict states) is ensured by randomly picking the examination times. Because the time between examinations usually includes only a few changes of state, randomness can only be ensured in this way. In computer sampling, the intervals between samples are long relative to the speed of CPU operations (usually on the order of 17,000 to 1 or 2 μsec); the system is assumed to have dramatically and unpredictably changed states between samples. Therefore, in computer sampling, randomness is obtained through the large and unknown number of changes of state between samples rather than through random timing of examinations. Instances where this would have totally unrepresentative results can be easily imagined but appear highly unlikely in typical systems.

The durations of I/O device operations (disk seeks, tape reads, printing a buffer, etc.) are usually significantly longer than the time between samples. The monitor examines CPU status and the status of these devices. Examination of I/O equipment is more complex than the random sampling of CPU-related items. During an operation, all examinations find the device busy. At the beginning and end of an operation, a sample may find the device either busy or idle, depending randomly both on the precise time the CPU makes an I/O request and the time required to perform the operation. If I/O operation times were a constant value or I/O requests were made at particular times relative to the time of sample interrupts, randomness would not be present. However, these conditions clearly do not seem to hold;[†] we can thus safely assume random sampling.

Sampling techniques enable the systems analyst to collect representative data over an extended period with reduced artifact and small resource expenditure for data reduction. Sampled data, sometimes combined with trace data,[‡] often adequately fulfill most of the data requirements of systems analysts.

Special Programs

Various installations have implemented a variety of programs to examine special problems. One of the most common application areas is real-time and on-line systems. To be viable, most of these systems must be special-purpose; therefore, special monitoring techniques must be employed to obtain data.

For example, American Air Lines' SABRE system has a series of monitoring programs integrated in its code to collect utilization and response time values. The programs are activated at an on-line terminal. Most large systems have at least one special monitor of this type, but these are seldom documented.

[†]I/O equipment has mechanical components that have non-constant times for identical requests. In multiprogramming systems, I/O requests tend to be random because jobs switch control, disrupting the sequence for any particular job.

[‡]The monitor developed at SLAC uses both sampling and tracing. Most system functions are sampled, but important activities occurring while the system is masked are traced. All I/O operations are traced to identify their type and all supervisor calls are traced to identify the routine called and any user module that may be brought in. Measurements made with a sampling software monitor and a full trace hardware monitor on a 360/65 operating OS/MVT illustrate the importance of this masking. The sampling monitor indicated that the 360's CPU was in supervisor state 10 percent of the total time whereas the hardware monitor showed 25 percent. The two monitors indicated similar total CPU utilization, 56 and 57 percent.

CATEGORIZING MONITORS

Different groups of people design and implement hardware and software monitors. This difference in personnel is probably the main reason for the emphasis on distinguishing between hardware and software. However, this distinction is only one of the major tool characteristics. This section discussed a number of specific measurement tools; we now summarize a framework for categorizing both currently available and as yet unimplemented techniques.[†]

The tools fall into five binary categories: implementation medium, separability, sample portion, analysis concurrency, and data presentation technique (see Table 1).

Table 1

BASIC TOOL CATEGORIES

Category	Characteristics
Implementation medium	Hardware Software
Separability	Integrated Separated
Sample portion	Full-time monitoring Sampling
Analysis concurrency	Concurrent analysis Recorded data
Data presentation	Static statistic Time-related

The "implementation medium" is either hardware or software; both are well understood.

[†]This framework modifies and extends one proposed by Kenneth Kolence [26]. Kolence's categorization includes a distinction between passive and active monitoring.

The "separability" characteristic, either integrated or separated, is often ignored because only separate monitors are usually considered. However, integrated accounting systems are standard and integrated hardware monitors are occasionally available. Many system operations, particularly software ones, can be most effectively monitored with integrated tools; these should be integrated during system design. For example, American Airlines' SABRE system included integrated software tools in the design shortly after the effort began. Monitors integrated after design completion tend to require excessive overhead and cannot aid the designers in producing a better system.

The "sample portion" indicates what portion of data is sampled. Full-time monitoring (tracing) considers each operation in a certain class (for example, all I/O interrupts or all times when a channel is busy). Sampling monitors are those in which the sample portion is less than 100 percent of the operations of the monitored system.

Collected data are either analyzed on-line or recorded for later analysis. "Analysis concurrency" can therefore be either concurrent or nonconcurrent with data collection. Accounting systems, for example, typically analyze data on-line and record results at the end of a job. On the other hand, hardware monitors often record raw data on tape for later analysis.

Common practice in the industry is to provide some data presentation tools along with commercially available data collection tools. The "data presentation" technique becomes associated with the data collection tool. Many techniques are in use, but the manner of summarizing the data is critically important for many types of analysis. Data presentation tends to fall into two categories: (1) static statistics (means and counts are most common), and (2) time-related displays (e.g., histograms of usage over time). Data collection tools with very rudimentary presentation tools (e.g., most accounting systems) fall, by default, into the category of static statistics (mainly raw data).

Five binary categories were described; therefore, five binary digits can describe any of the data collection tools. Table 2 illustrates this for several tools. The analyst choosing a monitoring tool can use this description technique to pick the characteristics desired and then find or design the needed tool.

Table 2

DESCRIPTIONS OF TOOLS

Characteristic		Typical Accounting	Typical Hardware Monitor	Mini-Computer Monitor	Comcet, 360/85 370/165	SUPERMON[a]
Hardware	0	1	0	0	0	1
Software	1					
Integrated	0	0	1	1	0	1
Separated	1					
Full-time Monitoring	0	0	0	1	0	0,1
Sampling	1					
Concurrent Analysis	0	0	1	0	0	0
Recorded Data	1					
Static Statistics	0	0	0,1	0	0	0
Time-related	1					

[a]SUPERMON is described by its indicated characteristics and is for examining IBM 360 systems (see Ref. 3).

IV. BEYOND TOOLS: A CAVEAT

This report discussed some of the reasons for performance analysis and the tools for measuring performance. The decision to do performance analysis is certainly an important one, and the effort is futile without appropriate measurement tools. The implication may be that deciding to analyze performance and choosing tools constitute the bulk of the effort in this area. This implication is inaccurate, and poor analyses result when it is implicitly accepted as true. An analyst may decide measurement and analysis are important, choose a tool, collect some data, and then attempt immediate conclusions. Procedurally, there are more effective ways to go about performance analysis.†

Many important issues remain outside the scope of this report. For example, simply measuring--collecting a set of random values--is almost always a waste of time. Considerable preliminary work must go into developing performance models and testable hypotheses. This effort turns an amorphous "feeling" into a test to distinguish between fact and imagination.

Testing can be done either by monitoring normal performance or by running a controlled experiment [27]. Monitoring has the advantage of relevance to the situation but the disadvantage of spurious effects combining with the variables under study. Controlled experiments are seldom run except in the case of benchmarking to compare several systems or to track performance over time. A series of well-defined tests can yield data to test an impressive set of hypotheses, but the hypotheses may be irrelevant if normal performance is not also carefully monitored.

---

†See Ref. 1 for procedural suggestions on simulating performance in systems under development.

Designing controlled experiments and monitoring situations is always critically important. Because unanticipated variables always appear, experimental design is a continuing problem. In one series of controlled experiments at Rand, we found problems in benchmark jobs, difficulty in controlling variables, and an inherent variability in the measured process. Because we anticipated problems, we were able to learn from early runs and actually use much of the data. The final experimental design allowed us to analyze the data in a variety of ways so that some models hypothesized after the runs could be checked and discarded.

Raw data must be transformed into answers about hypotheses. A large body of general data analysis techniques exist that can be applied to analyzing performance data. These techniques must be specialized in order to test hypotheses about computer performance; over-simplified techniques are usually employed. A simple numerical comparison of two values of unknown reliability and relevance is often used to make major decisions. More sophisticated and reliable techniques are commonly known and applied in other fields; they should also be used in computer performance analysis.

Each of these four topics--modeling, data collection mode, experimental design, and data analysis--is probably more difficult and important than choosing data collection tools. The variety of available tools discussed in this report should indicate that good data collection does not dominate other problems. Devoting adequate effort to each of the five topics can appreciably increase the probability that computer performance analysis will achieve its objectives.

REFERENCES

1. Bell, T. E., *Modeling the Video Graphics System: Procedure and Model Description*, The Rand Corporation, R-519-PR, December 1970.

2. Bemer, R. W., and A. L. Ellison, "Software Instrumentation Systems for Optimum Performance," *Proc. IFIP Congress 1968*, pp. 39-42.

3. Johnston, T. Y., "Hardware Versus Software Monitors," *Proc. of SHARE XXXIV*, Vol. I, March 1970, pp. 523-547.

4. Saltzer, J. H., and J. W. Gintell, "The Instrumentation of Multics," *Comm. of ACM*, Vol. 13, No. 8, August 1970, pp. 495-500.

5. Wiener, J., and T. DeMarco, "The Systems Scene: Tuning for Performance," *Modern Data*, January 1970, p. 54.

6. Denning, P. J., "Thrashing: Its Causes and Prevention," *Proc. AFIPS 1968 Fall Joint Computer Conference*, Vol. 33, pp. 915-922.

7. Randell, B., "A Note on Storage Fragmentation and Program Segmentation," *Comm. of the ACM*, Vol. 12, No. 7, July 1969, pp. 365-369.

8. Blau, P. M., *The Dynamics of Bureaucracy*, The University of Chicago Press, Chicago, 1955.

9. *IBM Systems Measurement Instrument Service*, International Business Machines Corp., Data Processing Division, G520-2295, White Plains, New York [Brochure].

10. *System Utilization Report*, Computer Synectics, Inc., Santa Clara, California [Brochure].

11. *CPM II Computer Performance Monitor*, Allied Computer Technology, Inc., Heuristic Systems Division, Santa Monica, California [Brochure].

12. *X-RAY Computer Performance Measurement System*, Applied Systems Division, Computer Learning and Systems Corp., Chevy Chase, Maryland [Brochure].

13. *Series 7700 Computer Performance Analyzer*, Computer and Programming Analysis, Inc., Chevy Hill, New Jersey [Brochure].

14. Schulman, F. D., "Hardware Measurement Device for IBM System/360 Time Sharing Evaluation," *Proc. ACM 22nd National Conference*, 1967, pp. 103-109.

15. Rock, D. J., and W. C. Emerson, "A Hardware Instrumentation Approach to Evaluation of a Large Scale System," *Proc. of the 24th National Conference (ACM)*, 1969, pp. 351-367.

16. Estrin, G., D. Hopkins, B. Coggan, and S. D. Crocker, "SNUPER Computer: A Computer in Instrumentation Automaton," *Proc. AFIPS 1967 Spring Joint Computer Conference*, pp. 645-656.

17. Murphy, R. W., "The System Logic and Usage Recorder," *Proc. AFIPS 1969 Fall Joint Computer Conference*, pp. 219-229.

18. Zunich, L. H., "Study of OS 360 MVT System's CPU Timing," *SHARE Computer Measurement and Evaluation Newsletter*, No. 3, February 7, 1970.

19. Stevens, D. F., "On Overcoming High-Priority Paralysis in Multiprogramming Systems: A Case History," *Comm. of ACM*, Vol. 11, No. 8, August 1968, pp. 539-541.

20. *Advanced Multiprogramming Analysis Procedure (AMAP) Service Description Manual*, International Business Machines Corp., Data Processing Division, GH20-0725, White Plains, New York [Brochure].

21. Cantrell, H. N., and A. L. Ellison, "Multiprogramming System Performance Measurement and Analysis," *Proc. of the 1968 Spring Joint Computer Conference*, pp. 213-221.

22. *A Guide to Program Improvement with LEAP*, Lambda LEAP Office, Arlington, Virginia [Brochure].

23. *Systems Measurement Software (SMS/360) Configuration Utilization Efficiency (CUE) Product Description*, Boole and Babbage, Inc., Palo Alto, California [Brochure].

24. Kurnow, E., G. J. Glasser, and F. R. Ottman, *Statistics for Business Decision*, Richard D. Irwin, Homewood, Illinois, 1959, pp. 212ff.

25. Buffa, E. S., *Modern Production Management*, John Wiley & Sons, Inc., New York, 1965, pp. 744ff.

26. Kolence, K. W., "Systems Measurement--Theory and Practice," *Proc. of SHARE XXXIV*, Vol. 1, March 1970, pp. 510-521.

27. Karush, A. D., *Two Approaches for Measuring the Performance of Time-Sharing Systems*, System Development Corporation, SP-3364, Santa Monica, California, May 1969.