SLAC-PUB-695
November 1969
(MISC)

EVALUATION OF SYSTEMS PERFORMANCE IN A MULTI-PROGRAMMING ENVIRONMENT*

V. Androsciani
IBM, Data Processing Division

C. R. Dickens
Stanford Computation Center (SLAC Facility)

T. Y. Johnston
Stanford Computation Center (SLAC Facility)

R. Lonergan
IBM, Data Processing Division

T. Y. Johnston
Computer Facility
Stanford Linear Accelerator Center (SLAC)
Stanford University
2575 Sand Hill Road
Menlo Park, California 94305

(Submitted to the 1970 Spring Joint Computer Conference, Princeton, New Jersey 08540.)

*Work supported in part by the U. S. Atomic Energy Commission.

ABSTRACT

This paper examines five different system measurement techniques used at the Stanford Linear Accelerator Center with some notes on the history of their local development. An attempt is made to show the relative merit of each technique.

The techniques are:

1. Benchmarks
2. Individual program measurement
3. Analysis of systems accounting information and the use of charging
4. Hardware monitors
5. Software monitors

The need for global measurement, understanding, monitoring and changing of complex multi-programmed systems is emphasized. The techniques described are heuristic and iterative rather than analytical. When a multi-programming system can be described analytically, then perhaps analytic techniques can be used to improve performance. The continuously changeable and non-forecastable nature of complex multi-programming operating systems is examined. The need for comprehensive measurement techniques is espoused. Further, basic characteristics of measuring tools are discussed, including: ease of use, economy, availability and comprehensiveness.

I. INTRODUCTION

The Stanford Linear Accelerator Center (SLAC) is a laboratory performing basic research in high energy physics operated by Stanford University under contract to the Atomic Energy Commission. The major computer system at SLAC is an IBM 360, Model 91, operating under the multiple variable tasking option of IBM's Operating System. The goal at the Facility is to provide the best possible service for a wide range of users with diverse requirements. This paper, then, is an exposition of how evaluation of systems performance on the Model 91 is accomplished. The primary goal in measuring performance is to improve the utilization of this large processor.

As a secondary goal, understanding the basic elements which affect performance within such a complex system is also important so that, for example, a future system can be written which optimizes all important elements.

This paper is neither a survey of the field of performance measurement nor is it an analytic description of the discipline, rather, it is a description of how SLAC has and is going about the problem of understanding the dynamic operating and constantly changing characteristics of a multi-programming system. We feel that our experience and the development work that we have engaged in over the last two years is of interest to all members of the computing community from those who are attempting to run a multi-programming system to those who are producing both software and hardware components.

II. BASIC PRINCIPLES

The overriding principle and objective of systems evaluation is to understand the unknown (1). This pervades any discussion of systems measurement and analysis. Several practical rules of thumb and principles have evolved in our pursuit of

understanding. Learning is a continuing process hence systems performance evaluation is both a continuing process and a constantly changing process. This is complicated by the fact that what is being measured is a system to which changes are being made. As a result the study of systems performance is an evolutionary process using the results of today not only to improve performance but also to upgrade measurement and analysis techniques for tomorrow.

If indeed the evaluation of performance is evolutionary and of a continuous nature, the tools for measurement and analysis must be usable. The primary requisite of usability is availability. The tool that is here today and gone tomorrow is less than useful. Probably the most dangerous mistake one can make is to measure the performance of the system and extrapolate the results into the future without continued remeasurement. This denies both the dynamic and changing nature of operating systems.

For the same reasons mere availability is not sufficient. Usability certainly implies more than availability. Measurement and analysis tools must be easy to use. They should be economical and as simple as is consistent with the job at hand and the resources available. Evaluation of performance should not and need not consume large amounts of the resources being measured or require large expenditures of people time and effort. Another aspect of usability is timeliness. The higher the frequency of measurement and analysis the better. This is inconsistent, in the extreme, with economy. This is why the stress should be laid on simplicity. Given the choice of a simple tool that can be used daily versus a complex tool that can only be used monthly, take the simple tool and use it daily. We have found from experience that by developing the correct tools that this is not a compromise at all. Fairly comprehensive tools can be developed and used on an almost continuous basis. What is a comprehensive tool? Clearly if operating systems were well understood or if there were only a few phenomena or resources

to measure the question would admit a simple answer. This is just not the case in a true multi-programming environment. However, there are some obvious considerations. Local measurement and analysis is not global. The locally optimized system is not necessarily and indeed is almost never globally optimized. The most reckless and often made assumption is that the problem of poor performance in one component of the system must be solved by expanding or improving that same component. Poor hardware performance does not imply the acquisition of more or better hardware. Thus having tools which merely measure hardware activity not only reflects a reckless assumption but usually reinforces it.

The real problem then is to analyze systems performance and not individual component performance, yet we can only measure the many individual components of the system both software and hardware. The usability argument tells us that we can't measure everything, but fortunately for any given problem a reasonable solution can be obtained by measuring a proper subset. This subset will vary with the problem.

The ideal tool then is one which has many probes which not only measure individual component performance but which indicate time relationships among components. The addition and deletion of probes must be easy. This ideal tool must also run in the standard operating environment without a large perturbation to that environment. No one tool or approach exists. Thus simultaneous approaches must be made. From an understanding that it is systems and not component performance we are analyzing, we see that the various measuring tools must have cohesiveness. The relationships among the tools themselves must be examined.

The need for analysis and the tools of measurement are fairly independent of the goals of the analysis. Only the interpretation of measurement data will vary from system to system and from time to time. Obviously, components of

different origin will require unique probes for measuring but on the whole it has been easy to supply these probes to a properly designed measurement tool.

III. HISTORY OF PERFORMANCE EVALUATION AT SLAC

As the result of a joint research agreement between SLAC and IBM, a few individuals at SLAC, as a part of this research effort, were able to use an IBM Model 50 which was situated at the Stanford Linear Accelerator Center. The mode of operation of the Model 50 was that of a hands-on machine. The user was his own operator. He, as a consequence, had a global view of his problem, both from a programming and operational point of view. The need for optimization of individual programs soon became apparent. Since the Model 50 was running under the Primary Control Program, a sequential system, there were essentially no measurement tools available. The first ad hoc method of measurement was watching the IBM 2311 disk units vibrating. By watching the motion of these units, one could determine severe disk arm contention. With this extremely primitive tool access to data sets was improved. At the same time, a small transistor radio was hung next to the CPU, and by listening carefully one soon learned to distinguish prolonged activities of a specific kind. For example, channel program activity. These particular tools have long since been forgotten. During this same period, it was observed that watching the sequence of events denoted by the flashing lights on the console and control units provided even more information. It was not for another two years that this idea reached full fruition, although it was not uncommon to walk into the shop and find all the control unit doors open. Indeed these control units were placed so that the user could sit at the console, and observe the activity of the lights.

A primitive accounting program with manually logged and punched card input was written, however, at this time no idea for the use of this information for the

evaluation or measurement of performance was considered. It was merely a device to help in the gross allocation and scheduling of the resources.

With the advent of an IBM 360, Model 75, the first computer for SLAC's Central Computing Facility, the mode of operation for those who had been running on the Model 50, was completely changed. There was no more hands-on use. The programmer and the operator were now two separate individuals. There was, since this was very early in the life of a system, no formal effort to develop a capability for the measuring or evaluation of systems performance. Indeed it was over six months before even crude accounting data became available and at that, it was merely a by-product of the introduction of a HASP-MFTI System. The apparent need for developing performance evaluation tools had lapsed. Probably because the operator and programmer were now distinct individuals. Even the accounting data provided was used more to tell individuals "who had used what," rather than to determine how the system should be run or modified to improve performance.

In March of 1968 with the installation of an IBM Model 91 only a few months away, it was realized that sufficient quantitative information was not available to be able to precisely determine what gains the installation of the 91 would bring, given the same jobs that were running on the 75. Several benchmark programs had been run recently which produced very few real results. Discussions and talk began as to how to characterize the dynamic properties of the multiprogramming system soon to be used on the Model 91. The key word became quantification, not qualification. Every effort must be made to expand the knowledge of this system during normal operation. Where are we now over a year later? What have we done? The first idea is an exploitation going back to the original days of running the Model 50. Instead of just watching the lights on

the console a hardware monitor was built by SLAC to record their state. The hardware monitor was soon found to be wholly inadequate as was the use of benchmarks. The next method was to make better use of the accounting information. Thus even prior to the installation of the 91 with an MVT multi-programming system, there was an effort to try and develop techniques to capture and record as much information as possible about the running of an individual job in the system. Another obvious tool was to develop an external monitor for individual programs. The inadequacy of a hardware monitor and the other methods caused us finally to develop a software monitor which would augment and increase our capabilities for performance measurement. These then are the tools that have been used to evaluate performance of the system here at SLAC.

The five tools then are:

1. Benchmarks
2. Individual program optimization
3. System accounting and charging data
4. Hardware monitor
5. Software monitor

Each of these subjects will be discussed and the value of each weighed.

IV. BENCHMARK TESTS

Although the use of benchmarks has historically played a strong role in the selection of computer systems, it has also played a role in the continued evaluation of performance. Beginning in the latter half of 1967 and for about 8 months, the SLAC Facility ran a series of benchmark tests on the test Model 91 at Poughkeepsie. This effort amounted to a little over four man-months of systems programmer time. The overall objectives of these tests were: (1) to compare

the CPU of the Model 75 with that of the Model 91, (2) to gain an idea of the optimum operating procedures and optimum balance for the system that was to be installed later in 1968, and (3) to gain an understanding of the dynamics of a multi-programming system.

There were a total of 11 distinct jobs for the benchmark. One of the 11 jobs was a standard production program used here at SLAC on the Model 75. After four test sessions over the aforementioned period, the aggregate results were analyzed. It became apparent that the objective of comparing the CPU of the 75 against that of the 91, was fairly well met. A ratio of between 3 to 1 and 4 to 1 was established on this set of jobs constituting the benchmark, which we felt at that time was representative of our shop. This was the simplest test of all, running each job one at a time, sequentially, on the Model 91.

It soon became apparent that there were no conclusive results, whatsoever, as far as the other objectives were concerned. With more than one job in the 91 in the multi-programming environment, the test results varied as a function of the order in which the jobs were submitted, the way in which the data was distributed both across devices and channels, and seemed to be strongly dependent on the I/O activity of the individual jobs. Indeed, one peculiarity in using six identical jobs which were merely renamed showed up. The order of completion was independent of the order of submission. This was not understood at the time. One apparent conclusion that resulted from the series of tests was that no more than three jobs should be multi-programmed because overall throughput suffered. This has been found to be a false conclusion. The use of benchmarks for evaluating performance on a continuing basis has been essentially abandoned at SLAC as a reliable tool for the evaluation of our system, although in recent months we have cooperated with other AEC Facilities in running other benchmark tests. It must be

pointed out here that the weaknesses found in running the benchmark are: (1) in selecting the appropriate job mix, (2) in interpreting the data, both in terms of finding the causes of certain phenomena and of just processing the data gathered in a reasonable and timely fashion, and (3) it causes a severe displacement of the standard operation.

V. INDIVIDUAL PROGRAM OPTIMIZATION

In optimizing individual tasks the area that can be most readily improved is CPU utilization. It is also possible in the case of overlay programs to be able to examine these programs and find out if the overlay structure is properly constituted.

Modern operating systems allow extremely dynamic program structures, master tasks with the subtasks relating asynchronously to them. Subtasks and master tasks which consist of many separate programs which dynamically transfer control from one to another both horizontally and vertically as well as predefined overlay structures. Consequently, if individual programs are to be measured, the tool must be capable of measuring all of the valid program structures available under the operating system. Further, it should, if possible, be able to make these measurements without changing the normal running environment of the program. Any program should be able to be measured including vendor supplied packages such as compilers without modifying these programs. Further, the measurement must be easy and the results easy to interpret. These are, then, the goals that were established at the Linear Accelerator Center in setting up an individual program analysis routine. The original package created did not meet all of these criteria. Primarily, the ease of interpreting the results was less than satisfactory. Consequently, the package has been rewritten to provide readily readable histograms

of the results and to provide these facilities with a minimum of effort on the part of the user.

The total cost of designing, writing, and implementing this package has been less than two man-months of effort.

The attempt to optimize input/output is a difficult process, particularly the optimization of input/output by balancing across channels. It has been found from experience that most programs are better off competing with themselves than interacting randomly with other programs. Consequently, on such things as disk device allocation, multiple work files for a single program better fit on one pack than spread across the system where they will compete with the work files from other programs. Controlled tests were run that showed in worst case conditions (with spread data sets) running three jobs in parallel took longer than running the same three jobs serially. By making the jobs compete with themselves, rather than others, the average performance of the parallel jobs showed a 100 percent throughput improvement over the serial operation. That is, if the serial operation time was equal to 3T where T is the time for 1 job run serially, then the improved total parallel time was equal to 1.5 T. These tests were run using Assembler F which is the standard assembler program provided for the system. The assembler is an example of a highly I/O round program with multiple work files. This type of analysis does not necessarily hold true in the pseudo multi-programming environment where specific large tasks requiring a major portion of system resources run for long periods of time and have forecastable characteristics. The true multi-programming environment is not forecastable and consequently, attempting to balance input/output across devices is a hazardous undertaking.

The operating system required by the SLAC Package must have two capabilities: (1) a task-timing capability and (2) the ability for tasks to have subtasks running under them asynchronously. The package used at SLAC is timer driven. This has been found to be a valid technique and one where the measuring interval can be changed so that the amount of data created is not overwhelming and the data is readily processible by the second phase of the operation. It is necessary for such a timing-task to have output capabilities for putting out the information since, in-core summarization is too expensive in CPU time and space. The overhead from the timer program is extremely low and scarcely measurable.

It is possible with this type of technique to measure a program's performance in its normal running environment, not just in a stand-alone system environment. This will show, normally, a higher degree of WAIT, but will not distort appreciably the program's functioning. It shows how it really functions in the normal world. If an attempt is being made to optimize I/O, then the only valid technique is to run in the normal environment. Further, multiple tests should be run before and after any change to validate the change.

On programs which are CPU bound, this type of tool provides an excellent measure of which instructions are using the majority of the CPU time. In one case measured, the replacing of a few small subroutines in a large FORTRAN program by assembler language routines reduced the run time by about 35 to 40 percent.

VI. ACCOUNTING AND CHARGING

The system accounting data provides a wealth of information, as well as the necessary figures for charging the user and for operating the installation. It also, via the charging algorithms selected, is a technique for guiding the user.

The cost of implementing good accounting is a continuing cost. Change is inherent in the process. The more that is learned about the system, the more it becomes apparent that certain information is required and other information is not necessary. This decision cannot be made before the process is started. It is very difficult to give a cost for the accounting and charging algorithms that we have now built into our system. Most of the cost involved has been charged out to operations overhead rather than to the performance measurement information which is merely a byproduct. However, it is a very important byproduct.

Figure 13 shows some of the results of one program which analyzes the accounting information. This is an abbreviated comparison of two months of this year showing several of the main processors including the user production programs, and also the summary relationship for the two months. This information has enabled us to validate that system changes have in fact improved system performance. It has shown the continuing growth of PL/1 use over the period of time. By examining the information that is available in the entire report showing all programs used within the shop during the month, decisions can be made on the allocation of resources for making changes and on the selection of what processors or processes should be changed. Further, this report has shown, as summarized in Fig. 13, that CPU utilization per unit of elapsed time increased between April and August over 30 percent with a fairly constant job mix.

The Linear Accelerator Center has used external job classes to define jobs of varying characteristics. Two questions come out of this. (1) Does the user truly understand his job characteristics and properly establish job classes? (2) Does the computer facility fully understand these job characteristics and know what they are in defining the classes? The accounting data provides the raw material to answer these questions. By the use of the accounting data and a histogram creating program, answers have been provided to both questions. Yes, the user does basically understand his job and is able to properly characterize his work. Secondly, data processing management does know what this class of work looks like as a group, how well it uses the facility, and can have some feeling of the impact of changing the mix of classes within the facility.

Charging can be part of the optimization procedure. With an I/O bound system it is reasonable to charge heavily for I/O to make people attempt to improve their utilization of I/O. Conversely, on a highly CPU bound system, I/O should be charged out at a lower rate. Consequently, the choice of charging algorithms can be used to make the user tend toward desired utilization of the system. The installation that chooses to follow this type of technique must bear in mind, however, that they can also make the user go toward what is worst for the system. An important concept of charging which is omitted or forgotten by many is that charging is a twofold thing. You must charge not only for what a person uses but for what he prevents others from using. The program that uses a tremendous amount of CPU time should be charged for CPU time. The program that comes into core and uses practically nothing and sits in memory and uses memory for a long period of time should be charged for its preventing others from using other system resources. It will be noted that our charging formula as yet does not contain the charge for the amount of time that a program occupies an area of memory without using other major resources.

Figure 14 shows the tail sheet associated with each job showing both the items for which charges are currently made and the current charging formula.

Basically this formula is an adaption of McDonnel-Douglas's charging algorithm. The unit is 1 second of CPU time for an 150K region. This is a meaningful amount of time on a 360/91.

VII. HARDWARE MONITOR

The hardware monitor used at SLAC was built to our specifications. It is a small, simple monitor and our manufacturing cost was in the neighborhood of eleven hundred dollars. It consists of 5 probes, and two 32-bit counters. Associated with these is a clock pulse, and/or circuitry such that all the Boolean functions in the disjunctive normal form of all 5 probes are directly available. The monitor was used extensively and now is rarely used. Today it serves a primary purpose of validating the measurements made by the software monitor.

Hardware monitors as a class of measuring instruments have several advantages as well as disadvantages. The most obvious advantage is the fact that they are a zero overhead item. Associated with this, however, is the fact that they can only measure discrete resources. The number of measurements they can make is extremely limited. In all cases, they are counter units. They don't tell you anything about the who's that are inherent in many things. Since they measure hardware action only, they do not show the software interfaces that are so important in modern operating systems.

Further, to use the modern monitors with their many probes and their direct readout features would take a crew consisting of a hardware engineer who knew where to put the probes, a systems designer to say what should be probed,

and a couple of programmers to convert the data and create the reports. The reports are not available now. They are available later. It is not easy to change and if it is more than a simple monitor, like the one used at SLAC, then it is not possible to freely move it around the system and measure varying items. It should be noted, however, some of the new hardware monitors do have some extremely desirable features. They have the capability of taking their counters and putting them out at regular intervals, such that by the use of some programming efforts, it is possible to create graphs in time, showing the interaction on the system. But no where out of the hardware monitor comes a reason why such an action took place during that time. Why were channel loadings up, why was CPU utilization up, why was channel loading on a specific channel high, what disk drive was it? Even if the disk drive is known, is the reason for high utilization of that drive known? Can you go back and recreate? Perhaps you do, perhaps the world has changed and you do not recreate it. These are just a few of the reasons that a hardware monitor, in and of itself, is not a satisfactory tool. It is better than no tool, but is not a sufficient tool to know what is going on in a modern system.

Figures 1 through 12 show the outputs from a software monitor. This is at least to orders of magnitude more information than that provided by any available hardware monitor.

For hardware monitors to be of value, they should be used, the results checked, changes made, and the process iterated. A one time shot may clean up a few major problems which might have disappeared anyway in time and could introduce new ones,unless a check is made.

The hardware monitor has been useful for measuring basic operating system performance. This is a discrete resource and one which cannot be measured

by our software monitor. It is also an area where we, as users, can do very little to change the performance.

VIII. SOFTWARE MONITOR

Unquestionably the most important measurement tool development at the Linear Accelerator Center has been the software monitor. There is approximately one man-year's effort in this system and a very trivial amount required to run it. This has provided management and the systems programmer the ability to really know what's going on in the system. What is the cost and overhead? There is of course a cost in both overhead and biasing of the system. The monitor takes about one and one half percent of the user core and less than .1% of the available CPU cycles. It summarizes all information in memory and prints out a summary report. Consequently, it does not bias any I/O during its operation. Comparing it with other software monitoring systems it has the advantage that it is cheap and easy to run. It can be invoked by the operator or from the job stream. The length of time it is run can be selected when it is invoked. The results are immediately obtainable. Of what need is this immediate information? Sometimes, unimportant; other times, as shown later, it has made the monitor worthwhile for this alone.

Examination of the monitor shows that it provides many, many probes into the system. Virtually hundreds of different probes can be added, deleted or changed as is necessary. As the system changes the probes can be easily changed and immediate results are available. Furthermore, it not only gives quantitative information but also gives qualitative information such as what programs are being used, what resident members of the operating system were utilized, and how much each major resource was utilized. These are capabilities beyond the purview of any hardware monitor, no matter how sophisticated.

The software monitor provides a whole spectrum of capabilities. It provides the ability to balance the system in many different ways: balancing I/O between channels, putting different modules resident in memory, balancing the types of jobs that are run. It gives the person responsible for this the ability to choose the most advantageous alternative. Further, because of its low overhead cost it gives one the ability to measure and know that the choice was good; or if it was bad, to correct it. It can spot unusual activity and can be invoked rapidly so that when an unusual occurrance happens on the system the results can be obtained and corrective action taken in a matter of a few minutes. Most important it can be used to validate what has been done, to make certain that the improvements were, in fact, improvements, and, if not, too allow as many tries as are necessary to improve the system. The iterative nature of software measurement is one which has been brought home time and time again. For example, the modules made resident in the system were first established according to the list supplied by the vendor. After using the software monitor for a short period of time in its original primitive form, it was found that this was a bad list and it was improved and changed. When other changes were made to the operating system these echoed into other changes within this area of system structure such that we again modified this list to improve performance. There is now a list which is known to be good now but it is not expected that this list will hold forever as the optimal list for running the system. The 30 percent throughput improvement between April and August is in large part attributable to iterative measurement, system change and subsequent performance improvements. The measurements spotted bottlenecks and effort could be applied to alleviate the bottleneck. For instance, the standard operating system will not allow any tasks to initiate or terminate while a disk

pack is being mounted. This mount takes 3 to 5 minutes and SLAC has both many short jobs and many disk mounts. Consequently many jobs were locked out longer than they would have run. By improving the standard algorithm so that only jobs requiring disk mounts were locked out the overall throughput was improved about 10%. Express jobs run during the day where fast turnaround is highly desirable improved even more than the entire mix.

Included as an appendix to this paper are exhibits of all the displays available from the monitor. It would be too time consuming to cover all these in detail at this point and many of these are too specific toward the SLAC hardware configuration to be of general interest. However, they provide a clear and concise picture of the entire system's performance.

Chart 3 shows the channel usage during a ten-minute sample. The first portion of this figure shows by channel the amount of in-use time and free time. Free but queued means that although the channel is not busy there is a direct access seek going on on a device connected to the channel. In the case of channel 0, free but queued is meaningless for channel 0 is a multiplexor channel, which can neither be measured properly by the software monitor nor a hardware monitor. From this chart it is apparent that there is a tremendous channel load on channel 5 with only slightly over 2% free time. This will be examined further in one of the other charts to determine why the tremendous utilization of this channel. The second portion of the chart shows the amount of overlap time; that is, what percentage of the time the number of channels shown were busy. Busy means I/O transfer going on over the channel while free means nothing going on in the channel or beyond. That is, approximately 16% of the time none of the channels were busy, 44% one channel only was busy, 29% two channels were busy, and so on. The third portion shows the concurrency of operation between

sets of channels; that is, for 1.89% of the time channels two and three were both busy at the same time, whereas for 32.39% of the time, one or the other of these two channels were busy.

An examination of the direct access device utilization in Fig. 4 shows the reason for the tremendous activity on channel 5. Unit 537 shows an activity of 99.3%. This happens to be a systems volume and this heavy activity probably indicates that the decision (not yet implemented) to place a different portion of the operating system on drum will improve performance considerably. This figure portrays the activity on the disk packs and drums and gives the ability to properly balance data sets across the various packs and the packs across the various channels.

Figure 2 (the I/O interrupts by device) gives the ability to count the total number of interrupts and also to count interrupts of certain classes associated with the devices. This has been an extremely important tool in spotting the malfunctioning of hardware. For instance, another computer connected onto the main system was erroneously generating interrupts at the rate of approximately 9,000 per second. By running the monitor this tremendous surge of interrupts was counted, the reason for the system degradation could be ascertained and the device was put off line until it could be fixed.

Figure 6 shows the enqueue waits within the system. Any multiprogramming system must at times ensure that certain resources are serially reusuable. These resources are placed into queues and are enqueued upon. Extreme degradation can occur if multiple tasks enqueue exclusively upon the same resource. By observing the performance of the system it was found that an enqueue was done during the mounting of a disk drive which, as mentioned previously, held up all initiation and termination of job steps. The

subsequent change has had a tremendous performance improvement within our environment and a constant watch is kept for other lock-out conditions. These queues are purely software queues and a hardware monitor could not possibly measure them.

Figure 12 shows a summary of the information obtained. It gives a quick once-over of what was happening and in the middle of it is a section showing possible bottlenecks and why the system was not running as well as it might. By examination of this page it is usually quite simple to proceed directly to the proper detail section and find the reason for the tieup. Further, it shows the amount of CPU time available and not used. By integrating all of the reports together a good picture can be obtained of what went on in the system and how well the system performed in many categories.

The software monitor thus provides a twofold facility of an immediate answer to critical systems problems which are degrading the performance of the system and secondly, a planning tool for examining the system as it changes over time for optimizing systems performance. It has proved to be a valuable tool also because it is cheap to run and easy to run and the results are easy to interpret. Furthermore, large amounts of machine time are not necessary to process the results as is the case for most interrupt driven monitors. The interrupt driven monitor provides a wealth of information but such a mass of information that the data reduction problem becomes of itself a significant problem. Such interrupt driven tools may be valuable on occasions but the primary problem is that the cost of running is so high that management refrains from using this tool frequently. It is too expensive a tool, and consequently, one of the major benefits, the ability to use it and reuse it to learn a little more about the complex environment is lost.

IX. CONCLUSIONS

It should be apparent from the foregoing five examples of measuring techniques that no one of these can stand alone. A software monitor, perhaps, comes the closest to it. However it is still necessary to be able to validate the software monitor's performance, to examine and see which processors have heavy utilization, to be able to change these processors if necessary, and to be able to examine them and know what parts of them need to be changed rather than by some seat-of-the-pants estimate figuring that this is a good area to change. Further, it must be emphasized that once changes have been made, the results of these changes must be measured and the system re-examined to see what its new characteristics are. The purpose of all of these tools is to make better use of the facilities available rather than to extrapolate the current misuse of facilities into some other usually more expensive hardware configuration.

The man-effort spent in the measurement area at the Linear Accelerator Center has not been large. Certain of the efforts have been pioneering efforts from which others can benefit and at a much smaller cost. However, unless some effort is made, the manager will never be able to see what he is doing, why he is doing it, or how he is doing in a multi-programming environment. Just as the cost of both core and system overhead of the software monitor has been trivial on our system, so we believe the cost of maintaining a continuing program of measuring and integrating the results of measurement is trivial on any large scale system. It is not clear how large a system must be before a full-time person could be supported in this effort. Probably any system that is a true multi-programming system (that is a system which does not consist of one or two set applications multi-programmed, but a multijob streamed system)

can justify the cost of measuring that system. Unless, of course, the utilization of the system is so low as to give no recapture from improving systems performance. Unfortunately, few in the industry are in this somewhat enviable position.

X. GUIDELINES FOR SELECTION OF PERFORMANCE EVALUATION TOOLS

Once it has been decided that evaluation of systems performance is necessary, several questions should be asked when making decisions as to the selection of any particular measurement tool or set of tools. The first question, of course, is usability. Usability is a function of complexity, continued availability and cost. Is it inexpensive to use and how much can you expect to gain from its use? Does using the tool take up so many resources that you can no longer run jobs in a nearly normal fashion? How many individuals do you have to allocate full time in order to be able to use this tool? How long will you have it? If the tool is only available for a few days, clearly it is of little value since these tools must be used on a continuing basis. Are the results timely? Do you get them back in short order or do you have to wait several days or even weeks for the interpreted data?

How does this tool relate to others? Can the data obtained from using this tool be related to the other tools in your repertoire? Does it measure what you want it to measure? If you are interested in knowing which particular data set should reside in a fast medium of storage — a hardware monitor may be of no value whatsoever.

Is this tool easily modified to adapt to your changing needs? When you have successfully measured performance for a period of time, your recognized needs for measurement increase. The possibilities are practically unlimited.

The tool, which is difficult to modify or adapt to increased or new needs, is one that may be of little utility in the future. Does it measure systems performance or just hardware performance? The tool must measure globally and not just suggest only one solution when there are two.

XI. WHERE DO WE GO FROM HERE?

Our future at SLAC in the evaluation of systems performance is one of continued effort in the areas already described. With modifications and additions to the system, there will always be a need for increased and modified capabilities. In particular, greater attention will be diverted to obtaining a better understanding of the resource dynamics of interactive systems. Certainly other facilities will find the need and the desire to follow the same path that we have followed.

As far as the computer industry is concerned there appears to be a growing market for these tools. Thus, we have seen increases in the last year or so in the marketing of performance evaluation tools. We hope that the industry will see fit to recognize the need for and provide integrated sets of tools, rather than individual tools marketed on an isolated basis, requiring the individual customer to either supplement or integrate other measurement tools.

Manufacturers of computer equipment can greatly ease the burden of measuring systems performance by providing the basic capabilities for gathering data for subsequent analysis. We have suffered, for example, because we are unable to measure the number of bytes transferred over a channel and the rate at which they are transferred. This should be simple enough to provide at the time of manufacture.

If indeed large numbers of facilities begin gathering and evaluating data, these results will not only be used by each individual facility, but we hope that

this data can be pooled in a common bank. The SHARE Reliability Survey, which covers the area of maintenance for 360's, is an indication of this. We think that mechanisms of this sort must develop so that the manufacturer, the computer designer, and the user will have a large pool of data from which to better understand individual instances of operating systems, and also by integrating, provide better systems design for tomorrow.

REFERENCES

1. H. N. Cantrell and A. L. Ellison, "Multiprogramming System Performance Measurement and Analysis," Proceedings, Spring Joint Computer Conference (1968).
2. R. Brody, "Model 91 Timing Tests," SLAC Systems Technical Memo No. 9, September 10, 1968.
3. J. Blackburn, J. VanderLans, "SLAC 360 Accounting Device," Systems Technical Memo No. 11, October 21, 1968.
4. C. Dickens, "Accounting Project," Systems Technical Memo No. 20, March 7, 1969.
5. J. Cromwell, "AMAP vs Boole and Babbage PPE/CUE," Systems Technical Memo No. 22, March 18, 1969.
6. C. Dickens, N. Nielsen, "Resource Allocation of the IBM 360, Model 91," Systems Technical Memo No. 25, July 3, 1969.
7. R. Lonergan, "LRL Berkeley Benchmark, Model 91 vs 6600," Systems Technical Memo No. 28, August 28, 1969.
8. V. Androsciani and R. Lonergan, "SUPERMON, a Software Monitor for Performance Evaluation," Systems Technical Memo No. 30 (in preparation).

SLAC Systems Technical Memos are rough-hewn internal documents which are available upon request.

SUPERVISOR CALL USAGE

NO.	USE	NO.	USE	NO.	USE
0	81,435	1	45,301	2	6,448
3	37,304	4	30,302	5	40
6	2,645	7	6,274	8	2,854
9	2,795	10	45,559	11	453
12	195	13	0	14	3,590
15	493	16	258	17	0
18	80	19	197	20	236
21	54	22	73	23	12
24	22	25	36	26	11
27	32	28	13	29	27
30	0	31	0	32	35
33	12	34	34	35	136
36	0	37	0	38	0
39	0	40	21	41	0
42	17	43	7	44	15
45	966	46	335	47	1,994
48	282	49	0	50	0
51	0	52	0	53	0
54	0	55	95	56	317
57	0	58	0	59	0
60	0	61	3	62	16
63	0	64	110	65	0
66	0	67	0	68	0
69	0	70	0	71	2,495
72	256	73	2	74	1
75	0	76	0	77	0
78	0	79	716	80	0
81	0	82	0	83	0
84	0	85	0	86	0
87	0	88	0	89	0
90	0	91	0	92	0
93	0	94	0	95	0
96	0	97	0	98	0
99	0	100	0	101	0
102	0	103	0	104	0
105	0	106	0	107	0
108	0	109	0	110	0
111	0	112	0	113	0
114	0	115	0	116	0
117	0	118	0	119	0
120	0	121	0	122	0
123	0	124	0	125	0
126	0	127	0	128	0
129	0	130	0	131	0
132	0	133	0	134	0
135	0	136	0	137	0
138	0	139	0	140	0
141	0	142	0	143	0
144	0	145	0	146	0
147	0	148	0	149	0
150	0	151	0	152	0
153	0	154	0	155	0
156	0	157	0	158	0
159	0	160	0	161	0
162	0	163	0	164	0
165	0	166	0	167	0

FIG. 1

I/O INTERRUPTS BY DEVICE

ADDRESS	INTERRUPTS	CSW'0080'0	CSW'0C0B'0	ADDRESS	INTERRUPTS	CSW'8080'0	CSW'0C0B'0
009	137	0	137	00C	0	0	0
00D	118	0	118	00E	1,354	0	1,354
00F	1,333	0	1,290	01C	767	0	763
01F	0	0	0	040	68	18	40
041	87	0	87	042	123	17	106
043	88	0	88	044	67	13	54
045	147	38	109	046	100	6	94
047	0	0	0	050	51	0	51
051	71	0	71	052	89	0	89
053	7	0	7	054	0	0	0
055	0	0	0	056	0	0	0
057	23	0	23	058	86	0	86
059	74	0	74	05A	4	0	4
05B	12	0	12	05C	0	0	0
05D	264	0	264	05E	0	0	0
05F	0	0	0	071	0	0	0
08F	0	0	0	087	0	0	0
08C	0	0	0	08D	0	0	0
08E	0	0	0	08F	0	0	0
0C0	0	0	0	0C1	0	0	0
0C2	0	0	0	0C3	0	0	0
0C4	593	0	593	0D0	202	18	184
0D1	0	0	0	0D2	5,759	2,379	3,380
0D3	0	0	0	0E0	0	0	0
0E1	0	0	0	0E2	0	0	0
0F3	0	0	0	0E4	7,548	0	7,548
0F3	0	0	0	130	0	0	0
131	0	0	0	132	0	0	0
133	0	0	0	134	0	0	0
135	0	0	0	136	0	0	0
137	0	0	0	1C0	1,961	0	1,961
1D0	0	0	0	230	757	0	350
231	0	0	0	232	156	0	156
233	7	0	7	234	30	22	8
235	309	0	309	236	4,236	0	4,236
237	1,861	839	1,029	2C0	1,122	12	1,110
3F6	12	2	10	3E7	0	0	0
430	377	0	0	431	842	0	842
432	2,759	0	2,759	433	1,405	0	1,405
434	0	0	0	435	38	0	88
436	50	0	50	437	3,195	140	3,056
4E8	0	0	0	4E9	0	0	0
530	1,249	0	1,235	531	9	0	9
532	0	0	0	533	10	0	10
534	222	69	153	535	233	0	233
536	798	514	284	537	13,366	6,047	7,345
540	0	0	0	541	0	0	0
542	0	0	0	543	0	0	0
544	0	0	0	545	0	0	0
546	0	0	0	547	0	0	0
560	0	0	0	561	0	0	0
562	0	0	0	563	0	0	0
564	0	0	0	565	0	0	0
566	0	0	0	567	0	0	0
568	0	0	0	569	0	0	0
56A	0	0	0	56B	0	0	0

FIG. 2

CHANNEL USAGE IN PERCENT OF AVAILABLE TIME

CHAN	IN USE	FREE	FREE+Q	BUSY	BUSY+Q
0	.00%	.00%	100.00%	.00%	.00%
1	4.65%	95.35%	.00%	3.79%	.86%
2	27.48%	64.25%	8.27%	20.59%	6.89%
3	6.80%	93.20%	.00%	6.80%	.00%
4	24.38%	64.17%	11.46%	15.76%	8.61%
5	67.44%	2.33%	30.23%	39.36%	28.08%
6	.00%	.00%	.00%	.00%	.00%

CHANNEL OVERLAP	TIME ACTIVE
0	16.88%
1	44.79%
2	29.54%
3	8.35%
4	.34%
5	.09%
6	.00%
7	.00%

CHANNEL CONCURRENCE

CHANNELS	ALL	ANY
2-3	1.89%	32.39%
4-5	17.40%	74.42%

FIG. 3

DIRECT ACCESS DEVICE UTILIZATION

	ADDRESS	SERIAL NO.	USE COUNT	ALLOCATED	NOT READY	CU BUSY	SEEK	DATA TRANS	
2301	1C0	SCF0V0	0 - 1	100.00%	.00%	.00%	.00%	4.65%	
	1D0		0 - 0	100.00%	100.00%	.00%	.00%	.00%	
	2C0	SCF0V1	3 - 3	100.00%	.00%	.00%	.00%	2.67%	
2314	130		0 - 0	.00%	.00%	.00%	.00%	.00%	
	131		0 - 0	.00%	.00%	.00%	.00%	.00%	
	132		0 - 0	.00%	.00%	.00%	.00%	.00%	
	133		0 - 0	.00%	.00%	.00%	.00%	.00%	
	134		0 - 0	.00%	.00%	.00%	.00%	.00%	
	135		0 - 0	.00%	.00%	.00%	.00%	.00%	
	136		0 - 0	.00%	.00%	.00%	.00%	.00%	
	137		0 - 0	.00%	.00%	.00%	.00%	.00%	
	230	SPOOL1	1 - 1	100.00%	.00%	.00%	.26%	.95%	.27
	231	SCFEV4	8 - 8	100.00%	.00%	.00%	.00%	.00%	
	232	SCFEV5	7 - 8	100.00%	.00%	.17%	.00%	1.03%	
	233	PUB001	1 - 1	100.00%	.00%	.00%	.00%	.09%	
	234	PUB002	1 - 2	100.00%	.00%	.00%	.00%	.09%	
	235	PUB003	1 - 3	100.00%	.00%	.09%	.09%	.86%	.10
	236	CRBE10	2 - 3	100.00%	.00%	.17%	8.27%	16.28%	.51
	237	SYSDV2	1 - 7	100.00%	.00%	.52%	2.07%	5.51%	.38
	430	SCF170	0 - 0	.00%	.00%	.00%	.00%	.00%	
	431	SPOOL2	1 - 1	100.00%	.00%	.00%	.86%	1.29%	.67
	432	SPOOL3	1 - 1	100.00%	.00%	.09%	6.55%	4.82%	1.36
	433	CRBE20	3 - 3	100.00%	.00%	.09%	1.21%	6.46%	.19
	434	ADA004	1 - 1	100.00%	.00%	.00%	.00%	.00%	
	435	ICWL03	0 - 1	51.34%	47.55%	.00%	.00%	.43%	
	436	ICWL01	0 - 2	1.72%	.00%	.00%	.00%	.26%	
	437	SCF0V3	2 -23	100.00%	.00%	1.03%	5.51%	10.94%	.50
	530	SPOOL4	1 - 1	100.00%	.00%	.00%	2.15%	1.89%	1.14
	531	SCFFV8	1 - 2	100.00%	.00%	.00%	.00%	.00%	
	532	SCFEV9	1 - 1	100.00%	.00%	.00%	.00%	.00%	
	533	SYSDV1	2 - 2	100.00%	.00%	.00%	.00%	.00%	
	534	CG0003	0 - 8	36.09%	.00%	.00%	1.21%	.17%	7.00
	535	FB2DSK	0 - 2	10.85%	.00%	.00%	.26%	1.03%	.25
	536	ADA003	7 - 7	100.00%	.00%	.00%	.52%	.95%	.55
	537	SCF0V2	33 -33	100.00%	.00%	.00%	33.16%	63.14%	.53
	540		0 - 0	.00%	.00%	.00%	.00%	.00%	
	541		0 - 0	.00%	.00%	.00%	.00%	.00%	
	542		0 - 0	.00%	.00%	.00%	.00%	.00%	
	543		0 - 0	.00%	.00%	.00%	.00%	.00%	
	544		0 - 0	.00%	.00%	.00%	.00%	.00%	
	545		0 - 0	.00%	.00%	.00%	.00%	.00%	
	546		0 - 0	.00%	.00%	.00%	.00%	.00%	
	547		0 - 0	.00%	.00%	.00%	.00%	.00%	

FIG. 4

NON-DA DEVICE UTILIZATION

	ADDRESS	ALLOCATED	NOT READY	CU BUSY	BUSY	DATA TRANS
TAPE	0B7	.00%	.00%	.00%	.00%	.00%
	0C0	.00%	.00%	.00%	.00%	.00%
	0C1	.00%	.00%	.00%	.00%	.00%
	0C2	.00%	.00%	.00%	.00%	.00%
	0C3	.00%	.00%	.00%	.00%	.00%
	0C4	27.56%	15.68%	.00%	.00%	.78%
	0E0	.00%	.00%	.00%	.00%	.00%
	0F1	.00%	.00%	.00%	.00%	.00%
	0E2	.00%	.00%	.00%	.00%	.00%
	0E3	.00%	.00%	.00%	.00%	.00%
	0E4	94.14%	33.85%	.00%	.09%	31.52%
U/R	009	100.00%	.00%	.00%	.00%	59.69%
	00C	.00%	.00%	.00%	.00%	.00%
	00D	.00%	.00%	.00%	.00%	.00%
	00E	100.00%	.00%	.00%	.00%	99.83%
	00F	.00%	.00%	.00%	.00%	.00%
	01C	.00%	.00%	.00%	.00%	.00%
	01E	.00%	.00%	.00%	.00%	.00%
	040	100.00%	.00%	.09%	.00%	4.31%
	041	100.00%	.00%	.00%	.00%	11.11%
	042	100.00%	.00%	.00%	.00%	11.54%
	043	100.00%	.00%	.00%	.00%	10.68%
	044	36.09%	.00%	.60%	.00%	1.38%
	045	100.00%	.00%	.00%	.00%	11.71%
	046	100.00%	.00%	.00%	.00%	11.46%
	047	.00%	.00%	.00%	.00%	.00%
	050	100.00%	.00%	.00%	.00%	99.66%
	051	100.00%	.00%	.00%	.00%	95.09%
	052	100.00%	.00%	.00%	.00%	91.99%
	053	100.00%	.00%	.00%	.00%	5.17%
	054	100.00%	.00%	.00%	.00%	100.00%
	055	100.00%	.00%	.00%	.00%	100.00%
	056	36.09%	.00%	.00%	.00%	.00%
	057	100.00%	.00%	.00%	.00%	97.93%
	058	100.00%	.00%	.00%	.00%	87.51%
	059	100.00%	.00%	.00%	.00%	86.99%
	05A	100.00%	.00%	.00%	.00%	100.00%
	05B	100.00%	.00%	.00%	.00%	97.76%
	05C	100.00%	.00%	.00%	.00%	100.00%
	05D	100.00%	.00%	.00%	.00%	80.88%
	05E	100.00%	.00%	.00%	.00%	100.00%
	05F	.00%	.00%	.00%	.00%	.00%
	071	.00%	.00%	.00%	.00%	.00%
	08F	100.00%	.00%	.00%	.00%	.00%
	0BC	.00%	.00%	.00%	.00%	.00%
	0BD	.00%	.00%	.00%	.00%	.00%
	0BE	.00%	.00%	.00%	.00%	.00%
	0BF	.00%	.00%	.00%	.00%	.00%
	0D0	100.00%	.00%	.00%	.00%	.60%
	0D1	.00%	.00%	.00%	.00%	.00%
	0D2	100.00%	.00%	.00%	.00%	5.34%
	0D3	.00%	.00%	.00%	.00%	.00%
	0F3	100.00%	.00%	.00%	.00%	.00%
	3E6	100.00%	.00%	.00%	.00%	6.80%
	3E7	.00%	.00%	.00%	.00%	.00%

FIG. 5

PERCENT OF TIME SPENT WAITING FOR ENQ QUEUES

NAME	TIME WAITING
# SYSDSN	.00%
SYS1	.00%
USER	.00%
USER.VXA	.00%
USER.VXA.LINKLIB	.00%
USER.VXA.MACLIB	.00%
SYSIEECT	.00%
SYSIEFSD	.00%
QM	.00%
Q1	.00%
Q2	.00%
Q3	.00%
Q4	.00%
Q5	.00%
Q6	.00%
Q7	.00%
SYSVTOC	.43%
OTHERS	.00%

FIG. 6

IDENTIFICATION OF FREE CORE NOT IN A REGION

LOWER BOUNDARY	TOTAL CORE	LARGEST BLOCK
0K	.00%	.00%
75K	12.93%	12.93%
150K	.00%	84.91%
225K	84.48%	.00%
300K	.43%	2.16%
375K	2.16%	.00%
450K	.00%	.00%
525K	.00%	.00%
600K	.00%	.00%
675K	.00%	.00%
750K	.00%	.00%
825K	.00%	.00%
900K	.00%	.00%
975K	.00%	.00%
1,050K	.00%	.00%
1,125K	.00%	.00%
1,200K	.00%	.00%
1,275K	.00%	.00%
1,350K	.00%	.00%
1,425K	.00%	.00%
1,500K	.00%	.00%
1,575K	.00%	.00%
1,650K	.00%	.00%
1,725K	.00%	.00%
1,800K	.00%	.00%
1,875K	.00%	.00%
1,950K	.00%	.00%
2,025K	.00%	.00%

NO.	REGIONS AVAILABLE 300K	150K
0	97.84%	12.93%
1	2.16%	84.91%
2	.00%	2.16%
3	.00%	.00%
4	.00%	.00%
5	.00%	.00%
6	.00%	.00%
MORE	.00%	.00%

LOWER BOUND	FRAGMENTED CORE 300K	150K
0K	.00%	.00%
75K	15.09%	99.57%
150K	.00%	.43%
225K	84.48%	.00%
300K	.43%	.00%
375K	.00%	.00%
450K	.00%	.00%
MORE	.00%	.00%

FIG. 7

LINK PACK AREA MODULE USAGE

NAME	NO OF USES	% OF TIME USED	NAME	NO OF USES	% OF TIME USED
IEECVOP1	0	.00%	IEECVOP2	0	.00%
IEEPALTR	0	.00%	IEEPLDSP	0	.00%
IEEPPRES	0	.00%	IEEPRTN	0	.00%
IEEVMNT1	0	.00%	IEEVSTRT	0	.00%
IEEVWILK	0	100.00%	IEFQINTZ	0	.00%
IEFQMSSS	0	100.00%	IEFSD102	14	40.95%
IEFSD105	4	1.29%	IEFSD263	16	100.00%
IEEVHA	0	100.00%	IGC0A05A	0	
*IGC0C01I	199		*IGC0002	238	
*IGC0002A	54		*IGC0002B	73	
*IGC0002C	12		*IGC0002E	36	
*IGC0002I	27		*IGC0003B	35	
*IGC0003D	34		*IGC0003E	138	
*IGC0005F	95		*IGC0006D	110	
*IGC0007A	2,495		*IGC0007B	525	
IGC0107B	136		IGC0403D	20	
IGC3107B	421		IGC3207B	405	
IGC3407B	146		IGC3507B	156	
IGG019AA	12	100.00%	IGG019AB	4	100.00%
IGG019AC	3	11.64%	IGG019AD	0	.00%
IGG019AG	0	.00%	IGG019AH	0	.00%
IGG019AI	10	100.00%	IGG019AJ	7	100.00%
IGG019AK	4	100.00%	IGG019AL	0	.00%
IGG019AQ	19	100.00%	IGG019AR	21	100.00%
IGG019BA	175	100.00%	IGG019BB	173	100.00%
IGG019BC	128	100.00%	IGG019BE	0	.00%
IGG019CB	0	.00%	IGG019CC	112	100.00%
IGG019CD	91	100.00%	IGG019CE	26	100.00%
IGG019CF	11	100.00%	IGG019CH	91	100.00%
IGG019CI	63	100.00%	IGG019CJ	10	100.00%
IGG019CL	31	100.00%	IGG0190I	223	
IGG0190L	267		IGG0190M	267	
IGG0190N	267		IGG0190S	267	
IGG0190Y	226		IGG0190Z	223	
IGG0191A	259		IGG0191B	206	
IGG0191D	143		IGG0191I	59	
IGG0191N	221		IGG01910	207	
IGG01911	37		IGG0200A	97	
IGG0200F	201		IGG0200G	226	
IGG0200Y	201		IGG0200Z	226	
IGG0201A	168		IGG0290A	27	
IGG0290B	27		IGG0290C	27	
IGG0290D	27		IGG0290E	27	
IGG0325B	35		IGG0325D	35	
IGG0325F	34		IGG0325G	34	
IGG0325H	35		IGG0550I	18	
IGG0550K	6		IGG0550N	33	
IGG0550Z	36				

FIG. 8

MODULE USE: LINK LOAD XCTL COUNTS TOTALED

NAME	NO OF USES	NAME	NO OF USES
ASMF	0	ASMG	2
ASMGASM	0	COMPRESS	0
CRB1503D	1	IEBCOMPR	0
IEBCOPY	1	IEBEDIT	0
IEBGENER	1	IEBPTPCH	0
IEBUPDAT	0	IEBUPDTE	0
IECQBFG1	2	IEEPSTRT	0
IEESD563	0	IEESD564	0
IEESD565	0	IEEVACTL	0
IEEVATT1	0	IEEVDNX1	0
IEEVDOR1	0	IEEVDSP1	0
IEEVICLR	0	IEEVLNKT	0
IEEVRCTL	0	IEEVTCTL	0
IEEVWAIT	0	IEE0503D	0
IEFBR14	0	IEFIRC	0
IEFQDELE	0	IEFQMDQ2	0
IEFQMNQ2	0	IEFQMRAW	0
IEFSD061	4	IEFSD062	15
IEFSD065	0	IEFSD070	0
IEFSD071	0	IEFSD078	7
IEFSD079	7	IEFSD085	0
IEFSD086	7	IEFSD087	0
IEFSD094	0	IEFSD104	14
IEFVGM1	0	IEFVHN	0
IEFVH1	0	IEFVM1	0
IEFWC000	15	IEFW21SD	15
IEFW41SD	0	IEFW42SD	0
IEF085SD	0	IEF086SD	0
IEHDASDR	1	IEHINITT	0
IEHIOSUP	0	IEHLIST	0
IEHMOVE	0	IEHPROGM	0
IEHUCSLD	0	IEKAA00	2
IEMAA	0	IERRC000	0
IEUASM	0	IEWL	0
IEWLF128	2	IEWLF880	0
IEWLOADR	0	IEWSZOVR	990
IEYFORT	0	IFCEREP0	0
IGC0A01C	0	IGC0CLC1	0
IGC0001C	0	IGC0002F	2
IGC0002G	0	IGC0101C	0
IGC0103E	2	IGC0105A	0
IGC0107A	0	IGC0201C	0
IGC0205A	0	IGC0301C	0
IGC0305A	0	IGC0401C	0
IGC0405A	0	IGC0503D	1
IGC0505A	0	IGC0605A	0
IGC0703D	0	IGC0705A	0
IGC0803D	10	IGC0805A	0
IGC1103D	0	IGC1203D	2
IGC3607B	4	IGC3807B	0
IGG0CLC1	13	IGG0CLC2	0
IGG0CLC3	0	IGG0CLF2	4
IGG019AV	4	IGG019KA	2
IGG019KE	2	IGG019KK	2
IGG019KU	2	IGG019LI	2
IGG0190A	2	IGG0190B	2
IGG0190E	2	IGG0190J	2
IGG0190K	2	IGG019WB	0
IGG0190A	4	IGG0190D	4
IGG0190F	3	IGG0190J	2
IGG0190K	1	IGG0190P	1
IGG0190R	0	IGG0190T	4
IGG0190W	0	IGG0191C	4
IGG0191G	46	IGG0191J	19
IGG0191Q	0	IGG0193A	2
IGG0193C	2	IGG0193E	2
IGG0193Y	2	IGG0193Z	2
IGG02G0B	4	IGG0200C	3
IGG0201B	0	IGG0203A	2
IGG0203Y	1	IGG0230D	6
IGG0325C	0	IGG0550B	2
IGG0550F	0	IGG0550J	0
IGG0550L	3	IGG0550M	3
IGG0550U	0	IGG0550Y	0
IGG0551A	3	IGG0553A	3
IGG0553B	3	IGG0553C	3
IGG0553D	3	IGG0553E	3
IHEITGA	8	IHEOPQA	11
LINKEDIT	3	LOADER	0
MAIN	4		

FIG. 9

FIRST 20 MODULES USED BUT NOT IN LIST

NAME	NO. OF USES
MONITOR	0
IHEESMA	1,593
IHEEREA	1,593
IHEOPNA	10
IHEOPOA	10
IHEOPPA	10
IHECLTA	10
IGG0550A	2
SORT	4
IHECLTB	1
IGG0191K	4
IEHDASDS	1
IEHDPRNT	5
IEHDSCAN	10
IEHDDUMP	1
IGG019P8	1
IEHDPASS	1
IEHDAOUT	1
IEHDMSGB	2
SUPV	1
IGG0230C	1

FIG. 10

MODULE USAGE BY GROUPS.. NOT IN OTHER REPORTS

IDENTIFIER	COUNT
I	232
IEE	0
IEF	0
IEM	0
IER	232
IEU	0
IEY	0
IFF	0
OTHERS	18

FIG. 11

```
        MACHINE ACTIVITY AT A GLANCE
              DATE: 69.301
             ENDED: 16.27.20
    TIME MONITORED:  10.00 MINUTES


                  PARAMETERS
CYCLE RANGE
  CORE                5
  MODULES             5
  QUEUES              5
  I/O DEVICES         1
  CHANNELS            1
CYCLE TIME     0.50 SECONDS
CYCLES COMPLETED  1,161 OUT OF  1,200


                  ACTIVITY
CPU UTILIZATION                          32.88%
ANY SELECTOR CHANNEL BUSY                83.12%
I/O ACTIVITY INDEX                     64,651
I/O INTERRUPTS                         54,226       5,423 PER MINUTE
DEVICES USED                               50
RQE USE SINCE LAST IPL                     56
TOTAL SUPERVISOR CALLS                274,743      27,474 PER MINUTE
  EXCP                                 81,435       8,144 PER MINUTE
  WTO                                     136          14 PER MINUTE
  OPEN                                    267          27 PER MINUTE


              POSSIBLE BOTTLENECKS
ENQ WAITS                                .43%
300K REGION AVAILABLE                   2.16%
AVERAGE CORE WASTED                      203K
TAPE CU WAITING                          .00%
DISK CU WAITING                         2.15%
TAPE NOT READY                          4.95 MINUTES
DISK NOT READY                          4.75 MINUTES


                PROGRAMS USED
PROCESSORS
          ASSEMBLY      2
          COBOL         0
          FORTRANG      0
          FORTRANH      2
          LINKEDIT      5
          LOADER        0
          MAIN          4
          PL/1          0
          SORT          0
          UTILITY       3
STEPS INITIATED                          16


END OF MONITOR RUN --- VERSION 1.2 10/27/69
```

FIG. 12

PROCESSOR USAGE

SELECTED PROCESSOR	APRIL 1969 ELAPSED TIME CPU TIME RATIO	% CPU	% ELAPSED	NUMBER CALLS	AUGUST 1969 ELAPSED TIME CPU TIME RATIO	% CPU	% ELAPSED	NUMBER CALLS
UTILITIES	337/1	.07	3.6	1201	154.0	.07	2.0	1318
FORTRAN	14/1	3.6	6.8	5260	11.5	3.3	7.3	6869
PL1	24/1	.6	1.8	1089	15.7	.8	2.4	2155
LINKEDIT	93/1	.8	9.5	7830	64/1	.7	8.6	9765
PRODUCTION	4.3/1	87.8	50.8	6890	3.4/1	84.0	54.0	8595
TOTAL ALL JOBS	6.98/1			33000	5.33/1			35000

RATIO AUGUST/APRIL ELAPSED TIME 1.1/1 , CPU TIME 1.45/1

THROUGHPUT INCREASE FOR AUGUST OVER APRIL: 31%.

FIG. 13

```
                                JOB STATISTICS
30 OCT 69  TYJXX036  GP=SFSY  SEQ NO=206  CLASS=E  PRTY=12
*******************************************************************************
CLOCK TIMES (HH:MM:SS)
ON RDR     11:52:36  BEGIN EXEC  11:54:03  ON PRT     11:56:27
OFF RDR    11:52:37  END EXEC    11:54:43  OFF PRT    11:56:57
RDR TIME    0:00:00  EXEC TIME    0:00:40  PRT TIME    0:00:29
*******************************************************************************
RESOURCES USED          UNITS               CHARGE FORMULA
CARDS READ                79                R = MAX((N*0.075 - 300.0), 0)
CARDS PUNCHED              0
LINES PRINTED            290                 L = N * 3.75E-03
7-TRACK ACCESSES           0                 N7
          ERRORS        N.A.
          MOUNTS        N.A.                 M7
9-TRACK ACCESSES           0                 N9
          ERRORS        N.A.
          MOUNTS        N.A.                 M9
D.A.    ACCESSES         236                 ND
          MOUNTS        N.A.                 MD, M = (M7 + M9)*60 + MD*600
OTHER   ACCESSES           0                 NM, F = (N7+N9+ND+NM) * 0.075
CORE      AMOUNT        150K                 CF = N / 150
CORE  TIME  USED     0:00:30                 G = (8/9)*CF**2 - (2/3)*CF + (7/9)
CPU   TIME  USED     0:00:01.39              TSEC
SVC          WAIT        610
            OTHER      2,266 (NOT INCL I/O)
*******************************************************************************
TOTAL UNITS =           20.17                UNITS = G*(TSEC+F) + R + L + M
*******************************************************************************
```

FIG. 14