A historical summary of the growing complexities in computing systems and the effect of these complexities on system performance evaluation are presented.

The paper outlines basic approaches. It considers the application of test results and the test data itself. Two general approaches to gathering performance data are discussed.

A perspective on system performance evaluation

by M. E. Drummond, Jr.

Although digital computers are entering their third decade of existence, system performance evaluation techniques have been lagging by almost a decade. In the early days, computers were designed with a fairly precise objective in mind—make them as fast as possible, with acceptable reliability. However, the way that speed was measured depended on the application area. At that time, general-purpose computers were usually classified as either scientific or commercial, everything else being special purpose.

Scientific computers were judged by the speed of certain discrete capabilities, such as add, multiply, or divide time, since their principal application was to perform calculations. In many instances, the proposed use of the system was a well-defined application requiring a repetitive set of calculations, which was considered to be the main function of the calculation process. The user could then weigh the arithmetic speed of a system in relation to his calculation process.

Underlying this approach to judging a computer was the tacit assumption that arithmetic processing was its most time-consuming activity. Other activities, such as transferring data between main storage and auxiliary storage, were assumed to occur such a small fraction of the time that they were not worth taking into consideration.

The commercial data processing field assessed computing systems from the opposite end of their capabilities—their input/output characteristics. The commercial environment was rooted in card processing techniques, where literally tons of data had to be passed through the processing equipment to produce

payrolls, inventories, accounts, and billings. Thus, data for commercial computing systems were organized by unit records, and the main way of assessing a system's performance was by its record reading and writing rate.

computing system developments

Over the years, numerous changes to computing systems and their applications have combined to cause grossly erroneous results when overly simple evaluation techniques are used.[1]

An early change was the introduction of I/O buffering, which requires the analyst to consider overlap factors in his evaluation. Computer system performance was also affected by the development of large-capacity, direct-access storage devices, which allow the use of nonsequential access methods. Facilities that have been added to computer systems have frequently been in programming packages. The earliest and simplest facility was a programming package to load other programs from some input storage medium into main storage in the format required by the central processing unit. Then there were the basic utility routines to aid the programmer. The earliest types were the dump routines, for punching or printing the contents of the computer's main storage, and binary-decimal conversion routines. Around the same time, the symbolic assembly came into being. Provision was made in symbolic assembly programs for subroutines. These widely used instruction sequences (often collected together into a library) were arranged so that many different programs could use them. Relocatable loaders were developed so that subroutines could be loaded anywhere in main storage and referred to by the host program.

Perhaps one of the most important contributions to the field of computer science was the compiler, which allows the writing of programs in a source language that is reasonably close to the natural language for the application. Libraries of complete processing programs were introduced with utility programs to locate a program in auxiliary storage and relocatably load it into main storage.

Later, the various components (compilers, loaders, etc.) were joined together into a programming system package with job control facilities, which allow the user to specify the sequence of steps and complete jobs that he wants performed automatically. The job-to-job control provides a facility to assist operations personnel, rather than programmers, making computer systems easier to operate and therefore more efficient. Facilities have been added to the programming system packages to allocate computing system resources for the execution of many different jobs concurrently. Not only does this include batch processing, but also communications-oriented operations, such as remote inquiry or interactive computing.

Thus, as computing systems have become more complex, the analyst has had to take into consideration many new elements in evaluating systems. The operating system functions take computing system time that could otherwise be spent performing

mathematical or data processing work. But the objective of an operating system is to perform the main jobs more efficiently and hence more economically. The tradeoff is computing system capability against manual or semimanual procedures.

In setting the scene for the remaining papers in this issue, this paper outlines the basic approaches used to judge computer performance. The application of test results to actual situations is considered next, followed by a discussion of test data. Finally, the two basic methods for acquiring the performance data are considered.

Classes of evaluation

Performance calculations can be placed into either of two primary classes—availability or work capability. Availability expresses how much of the time a system (or part of a system) is, or can be, used for productive purposes. Work capability is an assessment of a system's ability and efficiency in performing an intended function.

availability

Availability may be expressed in absolute terms or as a percentage. In absolute terms, it is usually called good available time, which is the total (power-on) time less the maintenance time. However, in practice, an installation manager would consider many more factors. For example, he would be concerned with the distribution of total maintenance time into scheduled and unscheduled maintenance times.

Scheduled maintenance includes: the time to repair units that were previously determined to require repair but are not critical to the operation; preventive maintenance on those units that have a predictable failure rate based on previous history; and scheduled updating of the system to improve its performance. This work can reasonably be scheduled ahead of time. Unscheduled maintenance time, on the other hand, is that time during which the system must be repaired, because it cannot perform its intended function. There are three approaches to reducing unscheduled maintenance.

The first approach, building reliability into the various components of the system, is totally within the realm of the manufacturer. He must not only use highly reliable components, but also must recognize that some errors may be transient in nature. This latter factor influences the design of error correction and error retry schemes in system components.

The second method of reducing unscheduled maintenance, redundancy of critical units, is within the realm of the user. (This is analogous to the way in which a manufacturer improves reliability by using redundancy of circuits in the design of units.) Although we may think that redundancy is used principally in real-time systems, it is in fact quite popular with normal business applications. For example, at an installation in which a large number of magnetic tape drives is required at all times, one or

more additional units may be installed just to ensure that the necessary number of units are available when needed. This, of course, adds to the cost of the system, but may be economically justifiable in light of the work to be performed.

The third way to achieve higher availability involves both the manufacturer and the user. While a component necessary for part of the work is taken out of the system for maintenance, other work that does not need the unit is performed. In a way, it is analogous to rescheduling work around a unit that just failed. The primary difference is that the computer system itself does the rescheduling, performs the work that can be performed, and, if part of its capabilities are needed for the repair of the unit that failed, concurrently provides such service. Various names have been given to this type of operation, such as fail soft and graceful degradation. Evaluation of the availability of such systems can become extremely complex. Simulation techniques are generally used to predict their availability.

work capability

The calculation of work capability can take many forms. The three most popular measures of work capability are job time, throughput, and response time. Job time is a calculation of how long a system takes to perform an application. This criterion is usually applied to jobs such as sorting, compiling, or file updating. Throughput, which has a generic meaning that can be applied in a variety of circumstances, relates in some way to the rate of doing the total work of the system, rather than any single job. For example, if card processing rate is the critical parameter, a system's throughput may be expressed as a card rate. In a multiprogramming job shop environment, we are interested in jobs per day. A few years ago, throughput was used to indicate a relative performance factor between two systems. Because of the diversity of use, we present the phrase *relative system throughput*, which is defined later in the paper. Response time is usually expressed in absolute terms. But again, this is a phrase that requires further definition, depending on the context of the evaluation. In terminal-oriented systems, response time refers to the amount of time that the computing system takes to react to various transactions from the terminal. In other real-time systems, such as process control systems, response time can indicate the time needed to identify, load, and execute a critical function. Although no response to the activating source is required, there could be a requirement to finish some critical processing within a specified time. Response time calculations must be well defined within the context of their intended use.

Types of evaluation

The three primary types of evaluation are: classification, comparison, and time estimation.

classification

Classification, probably the most popular form of analysis, is seen in much of the literature on computing systems and is

also generally found in publications of companies that provide consultation on computing. There are many different classification schemes, ranging from systems based on a single attribute to complex formulas for determining a figure of merit.

One type of classification is the listing of all products in order of average (often assumed) purchase or rental price. Another type of classification is one based on some particular attribute, such as capacity of main storage, storage cycle time, or add time. In classifying systems by a single attribute, we quite often find them grouped in vague terms such as small, intermediate, and large systems, along with further qualification, such as very large, small intermediate, etc.

Classification by a single attribute can sometimes provide misleading information. Consider an intermediate computing system that has a processor storage cycle time of two microseconds and a small computing system that has a processor storage cycle time of 1.5 microseconds. A classification scheme based on storage cycle time alone would rank the small processor above the intermediate processor.

To overcome such deficiencies, techniques to provide *figures of merit* have been derived. For example, the classification can be based on *maximum storage bus rate* (MSBR):

$$\text{MSBR} = \frac{\text{data length}}{\text{storage cycle time}} \times \text{degree of interleave}$$

where data length is the total number of bits of information (including parity and control bits) accessed in one main storage cycle; storage cycle time is the time needed to read out from a physically identifiable storage unit one data length of information and to be ready to repeat the operation; and degree of interleave is a numerical value assigned to the ability of the system to have one or more physical storage units operating concurrently.

If we now reconsider the previous example, the small processor, which reads out nine bits of information in one storage cycle without interleaving, has a maximum storage bus rate

$$\text{MSBR}_1 = \frac{9}{1.5} \times 1 = 6 \text{ megabits per second}$$

and the intermediate processor, which accesses 36 bits in one cycle, has a

$$\text{MSBR}_2 = \frac{36}{2} \times 1 = 18 \text{ megabits per second}$$

Using this classification scheme, the intermediate system is ranked higher than the small system. Simply considering maximum storage bus rate in a comparative evaluation may be misleading, since it does not ensure that a system will use data at that rate. There could be peculiar uses of processor storage that do not allow the storage unit itself to operate at that rate. In addition, classi-

fication based on only a few attributes always involves the risk of ignoring attributes that significantly affect overall system performance.

comparative evaluation

When more than two systems are to be compared, one system is usually chosen as the base system, against which all others are evaluated. In producing a simple relative estimate, one makes the underlying assumption that the answers produced are indicative of relative performance of the systems being compared. In fact, performance may be influenced by many factors not taken into account in the comparison.

Comparative evaluations like other types often consider only the CPU and processor storage elements, with all auxiliary operations omitted. Two interdependent approaches have been developed: the *instruction mix* method and the *kernel* method. The methods differ more in the interpretation and subsequent use of the results than in their representation of any calculating or processing phenomena.

In the mix method, each instruction or related group of instructions in the repertoire of a computer is assigned a weighting factor obtained by analysis or measurement of a program or programs in execution. Applying the weight to each instruction provides an average instruction time that can form a basis of comparison between two or more systems. A major shortcoming of the mix approach is the use of a single set of weighting factors to assess the performance of systems with different instruction sets. In these circumstances, subjective judgment must be applied when using a mix approach.

The kernel method gets its name from the fact that the central or essential part of the application under study is examined. The general technique is to determine the most frequently used portions of an application and to program these portions in the various instruction sets of the central processing units being compared. A mix generally purports to represent a broader range of use than a kernel, because of the kernel's direct relationship to a single application, although it should be pointed out that some kernels are enormous in size, complexity, and analysis time. A kernel is structured after the scope of interest is determined. Usually a complex problem is broken down into a series of simple kernels for evaluation. After each kernel has been evaluated, they are recombined according to some weighting function, just as instruction times are combined in a mix process.

The kernel approach overcomes the deficiency of the mix approach between systems of different architecture, because the kernels are programmed or coded in the instruction sets of the various systems. Furthermore, the kernel retains the *sequence* of the instructions used. Although coding efficiency may affect performance, this approach provides reasonably accurate information for each kernel. However, it does take quite a bit of manpower to cover a broad range of applications. An example of the kernel approach was demonstrated by Hahn and Hankam.[2]

In comparing complete systems rather than merely central processing units or other individual devices, we use the concept of job time for the nonteleprocessing-oriented system. Relative system throughput (RST) is an estimate of performance of a computing system when measured against some base computing system. It is defined as the ratio of the time of computation for a given load on the base system (T_b) divided by the time of computation for the same load on the new system (T_n) or RST = T_b/T_n.

For this definition, the base computing system is an operational entity consisting of the interconnected components and devices of an electronic computer, a set of support programs (control program, compilers, etc.), and a set of procedures or application programs and the data processed by those programs. The set of procedures and associated data is usually called an environment. Naturally, the systems to be compared must have equivalent facilities, and the comparisons are no longer valid for a different environment.

Notice that the RST definition does not explicitly take into account two other measures of performance of a computing system—response time (or turnaround time) and availability. Relative system throughput is, therefore, an assessment of a system's job processing capability during the time that the jobs are under the influence and control of the system and the system is working.

absolute evaluation

An absolute evaluation is one that produces time estimates for the performance of a required function or operation. An absolute evaluation can serve either as a necessary step in the calculation of comparative performance or be the desired end product in itself. The techniques of obtaining absolute evaluations can vary depending on the accuracy required for the end use. A technique known to contain a consistent error, which produces some bias in the result, may be satisfactory for the intermediate calculations of relative system throughput, because of an assumption that the error will cancel out in the division. Also the data going into a relative evaluation may not be very accurate. On the other hand, if a specific job time is to be predicted, the tolerance for error may be very small. Given the end objective of the calculation, techniques should be chosen to fit requirements. An absolute evaluation obtained by simulative techniques was given by Baldwin, Gibson, and Poland.[3]

As an example of an absolute evaluation, consider the case of projecting compile time on a system. We can build up a job time from a calculation involving amount of CPU time required, amount of I/O time required, dependencies on data sets, and other relevant factors. Alternatively, we may take a more simplified approach. If we consider the act of compiling to be just another application and the compiler to be a known application program, we can express a timing formula for this application. Even though there are a large number of variables in the process of timing a com-

pilation (20 or so), we can calculate job time to satisfactory accuracy using a three-term equation. Therefore, assume that the time of each compiling job is expressed by the formula:

$$T_c = K + n \times R + p \times S$$

where T_c is time of compilation, K is a constant factor in the process of compiling, R is the time per source card of input, and S is the time per additional subprogram after the first. In evaluating the equation, n is the number of source cards in a program, and p is the number of subprograms after the first.

Therefore, calculating compile time on some new system would be merely the application of the compile time formula using coefficients relative to the new system with the data gathered (n and p for each compilation) from the present system. Summing all of the job times calculated for the new system, we may calculate the expected total compile time. It should be noted, however, that the result of the calculation indicates performance on only the application base of compiling. Furthermore, this simple equation does not take into account any effects of multiprogramming. It does provide a time estimate of a series of individual compilations.

Application of results

The preceding example would produce a single number as an expression of a system's merit. The next question is how can that result be applied to some other environment. To change an environmental or application characteristic requires a completely new calculation. When considering the almost infinite number of combinations of environmental data, we can foresee a never ending regimen of calculation. To overcome this problem, many approaches have been taken. Two approaches, which are more popular than the others, involve standard environments and average environments.

In considering the use of these two approaches, we can draw a parallel with the stock market. The Dow Jones average is a weighted average of a selected set of stocks. Statistically, this average is considered to be a reasonable measure of market activity. On the other hand, the New York Stock Exchange index actually averages all transactions that took place that day and is truly an average. We all realize, however, that neither of these indicators lets us know what happened to any particular portfolio. Furthermore, because of the difference in approach, it is possible to have one technique indicating that the market is up and another indicating that it is down. This does not mean that one is right and the other is wrong. It simply means that while there is general correlation for the complete market over a period of time, there are instances where there is divergence of results.

We have the same situation with regard to performance data. Although we may establish standard environments, it is always possible that some average environment produces results that

vary. Neither calculation may necessarily produce a result that is applicable to a particular system configuration in a particular environment.

In many cases, it may be more desirable to calculate a range of answers for plotting purposes. If a major attribute of interest can be isolated, it can be used as an independent variable for the projection of system performance. Consider an example in which a system component (physical device or program) has an influence on relative system throughput depending on the proportion of time spent on compilation and linkage editing. The effect can be plotted as RST versus the ratio of compile and link-edit time to total process time. Given such a plot, an analyst can then determine the relative system throughput for any particular environment. Many times, the shape of the curve itself may be more important than any particular point. If the region of interest is in the flat area of the curve, coarse information may be applied. If the region is in the steep portion of the curve, the probability of error and resulting deviation of the answer is greater.

Choice of data

In structuring an evaluation, we must not only consider the information gained from the output of the evaluation and the technique of the evaluation, but also the choice of the data used in the evaluation. In general, those calculations that are oriented to the evaluation of a specific application require data that in some way represent that application. On the other hand, calculations that are oriented to the "guideline" type of information ought to cover the extremes of the expected range of interest with sufficient intermediate points to allow approximation.

Consider the case where the prime coefficients of the application of compiling are to be calculated. A way of doing this would be to put together a set of jobs that spans the extremes of all of the known variables of the solution in such a way as to allow a regression analysis or some similar technique to be applied. In this case, a wide range of jobs would be structured.

Since there is a wide span of requirements for the data to be used in evaluation, the question of the use of actual applications versus artificial applications immediately comes to mind. An actual application is one that is being performed at some installation as a portion of the productive work of that installation. An artificial application is one that is structured to give the system loading effect of some real application, but does not even pretend to produce the true result of the real application. A word of caution should be noted. In attempting to obtain an actual application, we may in fact obtain many artificial attributes. To obtain an actual application, we must obtain not only the application program but also the data that goes with it. In some cases, we must also obtain all other procedures and data that may be in some stage of concurrent processing with the job under

study. To really obtain all of that information, we may in many cases face the requirement that capturing the data may involve up to 40 disk packs of information, as well as about 100 reels of tape, for just one day's observation.

A way of overcoming these data requirements is to use an application program known as a *benchmark*. A benchmark is a particular programmed procedure with some associated data chosen in such a way as to impart meaning to the originator of the benchmark. Two classical benchmark problems are matrix inversion for the scientifically oriented community and payroll gross-to-net calculation for the commercially oriented community. For other applications, benchmark programs may be developed to provide information of interest to the user.

An alternative to the actual application or benchmark is the synthetic program. The synthetic program must match in its principal attributes the attributes of the applications that it is purporting to either represent or span. Note that the synthetic program need not necessarily produce the results of a particular program. Parameters may be varied to allow a study of sensitive functions. In this way, the results of the evaluation can provide either tables or graphs that through simple interpolation provide expected performance for particular applications. An example of a synthetic job is provided by Buchholz in this issue.

Finally, there are many predictive techniques that do not require detailed information on how applications were processed in the base system. The principal requirement is to know the identity and incidence of use of the applications.

Data acquisition

Two classes of techniques for the acquisition of data for evaluation purposes are currently being used—software measurement and hardware measurement. Almost all data may be acquired using software techniques. The principal reasons for using hardware techniques are ease of installation, ease of use, and the ability to obtain data in a way that does not interfere with the work in process. On the other hand, there are many attributes that are much more easily obtained by software techniques, such as job identification, data set identification, origin of requests for facilities, and other data-dependent information.

Most software techniques intercept in some way the normal flow of programmed procedures to obtain the required information. The extent to which this intercept technique is used depends heavily on the type of information required. If, for example, we merely wish to determine the type of jobs being processed by the system, a relatively simple interception of the job scheduler is all that is required. On the other hand, if we wish to have a distribution of each execution of all types of instructions in some particular program, the measurement technique takes the form of the interpretive trace program, with the consequent degradation of

performance. An example of a software technique was given by Stanley and Hertel,[4] and another is provided in this issue by Stanley.

Hardware devices generally sense electronic signals in the host system. These signals when decoded by the monitor provide the instrument with the capability to determine what is going on inside the system. (See the paper in this issue by Bonner.)

Perhaps the most dramatic attribute of the hardware monitor is that it can obtain data reflecting the occurrence and duration of many events simultaneously. Where time-of-day calculations and time intervals provide information on a job as a whole, the monitor provides information concerning the activities of various system components while doing the job.

It is useful to group the monitors into two classes, summary type devices and dynamic type devices. The summary type device accumulates data on the occurrence and duration of particular events for the total amount of time or total number of occurrences during the period of observation. A summary type device disregards the sequence of events. An early device of this class is described in *Throughput Evaluations . . . .*[5] This report summarizes the use of such a device at the Western Data Processing Center of the University of California, Los Angeles, during the period of October 1961 to March 1962. On the other hand, the dynamic type device records when an event occurred. It may or may not also include the timing characteristics that are provided by the summary type device. In the general case, devices in primary use today tend to be of the dynamic type. Schulman has described such a device.[6]

Summary comment

Many options are available to the analyst in performing a particular evaluation. As we progress through the steps of an evaluation, we keep uppermost in our minds the question: "What problem is being solved?" In structuring a measurement or evaluation experiment, an outline is prepared. Elements of the outline include:

What decision is being made?
How should the results be interpreted?
What is the most meaningful presentation of data?
What sensitive elements should be examined?
What technique should be used?
What input is required for the technique?
How should (or can) the input data be obtained?

One further note to be considered is the fact that we are operating in a dynamic industry. A few short years ago, many installations produced exactly the same job in exactly the same way day in and day out. Even in these cases, the requirement for system optimization is evident.

Although the original application or procedure performed on the system may remain basically the same, the system does not. Continual evaluation is required to track the system's performance during its installed life.

CITED REFERENCES

1. W. Buchholz, "A selected bibliography on computer system performance evaluation," *Computer Group News* **2**, No. 8 (March 1969).
2. S. G. Hahn and E. V. Hankam, "Kernel analysis of elliptic partial differential equations," *IBM Systems Journal* **5**, No. 4, 248–270 (1966).
3. F. R. Baldwin, W. B. Gibson, and C. B. Poland, "A multiprocessing approach to a large computer system," *IBM Systems Journal* **1**, No. 1, 64–76 (September 1962).
4. W. I. Stanley and H. F. Hertel, "Statistics gathering and simulation for the Apollo real-time operating system," *IBM Systems Journal* **7**, No. 2, 85–102 (1968).
5. *Throughput Evaluation · · · IBM 7090/7094 Data Processing Systems*, © 1963 by International Business Machines Corporation.
6. F. D. Schulman, "Hardware measurement device for IBM SYSTEM/360 time sharing evaluation," *Proceedings of the 22nd National Conference of the Association for Computing Machinery* **P-67**, 103–109 (August 1967).