

A Measurement Procedure for Queueing Network Models of Computer Systems

CLIFFORD A. ROSE

*Naval Electronic Systems Command, Washington, D.C. 20360**

This tutorial paper describes a procedure for obtaining input parameter values and output performance measures for a popular class of queueing network models. The procedure makes use of current measurement monitors as much as possible. We survey the two basic approaches to monitoring computer systems (event trace and sampling) and the three types of monitors (hardware, software, and hybrid). Also surveyed are measurement tools for the analytical modeling of several current families of computer systems. We discuss in detail examples of model validations and performance predictions to illustrate the measurement procedures and the class of models

Keywords and Phrases: queueing network models, analytical models, model validations, performance evaluation, measurements, monitors.

CR Categories: 8.1, 4.3

INTRODUCTION

In the last three to four years, there has been increased emphasis on the validation of queueing network models (QNMs) and a study of their ranges of applicability. Validations are described in several papers [BASK72, GIAM76, KIEN77, LIPS77, MOOR71, and SEKI72]. Various studies of ranges of applicability have been reported by others [BHAN74, BUZE75, DIET77, HUGH73, PRIC76, REIS74, ROSE77, and BUZE77].

QNMs are becoming a more widely used method for analyzing computer systems; however, they have not been applied in many potentially useful cases by data processing personnel in the field. Discussions with computer systems personnel indicate that there are two primary reasons for this situation.

- 1) System engineers and system programmers who are familiar with the detailed procedures of the computer's operating system and devices sometimes believe that the behavior of a computer is too complex to be captured by a QNM (or an analytical model in general).
- 2) It is often not apparent to the personnel in data processing centers how to obtain the input parameters for these QNMs.

Thus, some personnel in the field who could benefit from application of a QNM instead perceive them to be more of academic interest than a practical and useful tool. This paper will propose a set of measurement procedures for obtaining the input parameter values and output performance measures of QNMs. These procedures are oriented toward a popular class of QNMs and utilize to the maximum extent possible the measurement parameters that are commonly provided by current measurement monitors.

* The author is now at the Naval Sea Systems Command, Washington, D. C. 20360.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

CONTENTS

INTRODUCTION

1 MEASUREMENT MONITORS

Policies

Implementations

2 EXAMPLES OF SPECIFIC MEASUREMENT MONITORS AND SYSTEM ARCHITECTURES

IBM 370 Series

Honeywell 6000 Series

Univac 1110 Series

DEC 20 and 11/70 Computers

CDC 6600 Series

3 DESCRIPTION OF A QUEUEING NETWORK MODEL

4 MEASUREMENT PROCEDURES FOR OBTAINING QUEUEING NETWORK MODEL INPUT PARAMETERS

Number of Classes of Jobs

CPU Queueing Discipline

Degree of Multiprogramming

I/O Device Routing Frequencies

CPU Mean Service Time

I/O Device Mean Service Time

Optional Parameters

5 MODEL VALIDATIONS

6 PERFORMANCE PREDICTIONS

SUMMARY AND CONCLUSIONS

ACKNOWLEDGMENTS

REFERENCES

The organization of this paper is as follows. Section 1 will discuss types of measurement instrumentation that are of interest to the computer systems analyst, and Section 2 will present several measurement monitors available on some of the more popular families of computers. Section 3 will describe the baseline QNM, stating the basis for its selection and the measurement requirements. Section 4 will propose a measurement procedure for satisfying the requirements of this particular QNM. There will also be a discussion of the applicability of this procedure for two extensions of the model. Sections 5 and 6 present examples of model validations and performance predictions, which illustrate the measurement procedures and the class of models.

1. MEASUREMENT MONITORS

Measurement monitors are devices that observe the operation of a computer system over a period of time and record the values of certain variables that are considered to

be significant for evaluating system operation. For a description of performance measurement methods and technology, see [LUCA71 and SVOB76]. It will prove helpful at this point to distinguish between the types of *policies* and the types of *implementations* for measurement monitors. There are two types of policies, or high level designs, for monitors: event trace and sampling. There are three types of implementations: hardware, software, and hybrid. Each type of policy and implementation will be discussed in the following subsections.

Policies

Both event trace and sampling monitors record system states from specified registers and memory locations. They differ in the way they are invoked. An event trace monitor records information at the time of a particular event; a sampling monitor records information at specified time intervals.

The event trace mode usually obtains more detailed information about system operation over a shorter period of time. At the occurrence of a prescribed event, control of the computer operating system is passed to the event trace monitor. Typical events that could trigger data collection activities include start of a job, task creation, main storage allocation, I/O activity, supervisor interrupt, and job completion. The event trace monitor will then record data in buffers that can be written to disk or tape for subsequent use by data reduction software. Data recorded consists of information in predefined register and memory locations that are of interest at the time of occurrence of the particular event. For example, at time of job completion the event trace monitor might record the amount of CPU time used by the job, the number of page faults initiated, the number of I/O operations completed, and the number of tape mounts required. In addition to the predefined data to be recorded, most monitors permit the user to specify other information for collection.

Sampling monitors initiate data collection activities when a real-time clock signals the end of an interval. The interval, or sampling period, is usually constant. This

type of monitor usually provides a less detailed description of system operation over a longer period of time. Information provided by sampling monitors typically include CPU and channel utilizations and the number of jobs in execution. To obtain these measurements, the monitor would sample at prescribed intervals the state of the processors and the number of jobs in execution. At the completion of the monitoring session, the data reduction routines for the monitor would typically provide the percentage of time that the CPU and the channels were in the busy state, and a table or graphic plot of the number of jobs executing as a function of time during the test interval.

Implementations

There are three basic implementations of measurement monitors for computer systems. Each of the three types of monitors may use either event trace or sampling policies. *Hardware monitors* are devices that are attached to various points in the digital circuitry to measure status or to count events. *Software monitors* are measurement routines inserted into the system software that also record prescribed status and events. *Hybrid monitors* are combinations of these two types. Measurement information for hybrid monitors is moved by system software routines into special monitoring registers, and is then measured by hardware monitors.

A hardware monitor measures the state of the computer via electronic sensors, or probes, which are connected to certain points in the digital circuitry of the computer. The usual measurements are to count specified events or to record the elapsed time that the system is in a certain state. For example, a hardware monitor can be used to measure the amount of time that the CPU was in a busy state, or to count the number of times that a referenced memory address was in the high-speed cache memory. Since the hardware monitor directly measures the status of registers and memory locations, system resources (CPU and I/O processors) are not diverted from the workload processing. Thus an important advantage of hardware monitors is

that overhead is not introduced into the measurement process.

A typical hardware monitor will consist of between 20 and 150 probes. It will also include logic modules for performing Boolean operations such as AND and OR, a collection and distribution plugboard to count, time, and store the measurements as defined by the users, and a controller to coordinate and control the data collection functions. Many hardware monitors provide data reduction software for immediately reducing, tabulating, and displaying the results. Modern high performance hardware monitors may include a wide range of peripherals, such as disks, printers, and graphic displays, all under the control of a minicomputer. The analyst would interface with the older hardware monitors through switches, buttons, or a digital display. Current models feature keyboards, printers, and CRT consoles.

Software monitors in general can record any information that is available to (or contained within) an application program or the operating system. The "probes" for a software monitor consist of dedicated code that is executed for the purpose of recording the desired information. This code might be called periodically (sampling mode) or at the occurrence of a particular event (event trace). Access to both the operating system and the application programs provides a great deal of flexibility in the selection of the measurement data. For example, a software monitor would be able to access and record interesting information accumulated by the operating system, including resource usage and system tables.

The penalty for this increased flexibility is the perturbation to system operation. Execution of this code will place additional demands on the system resources that are being measured, e.g., the CPU, peripheral devices, and memory. The analyst should insure that the monitor does not introduce excessive overhead and render the results meaningless. For certain applications event trace monitors can record extensive amounts of data, and in these instances these monitors have been observed to introduce as much as 20% overhead. Reasonable selection of the events and the quan-

tity of data to be recorded for QNM applications should result in the amount of overhead caused by event trace monitors to be limited to approximately 5 to 10%.

The overhead imposed by the software sampling monitor is controlled by specifying the sampling frequency, with low sampling frequencies resulting in less overhead. On the other hand, the laws of statistical sampling apply, so the number of samples should be large enough to achieve a reasonable accuracy for the parameters of interest. Therefore what is needed is a sampling frequency for which the monitor can collect a representative sample of system operation without significantly affecting system performance.

Another factor affecting the accuracy of software sampling monitors is the priority of the monitor. The higher the priority of the sampling monitor, the less likely that it will be locked out at sampling time and the more accurate will be the results. The software sampling monitor for the 370/OS system in [ROSE75] executed as the highest priority system task and software monitor measurements were within 3% of hardware monitor readings. For the 370/VS systems the monitor executed with the second highest priority (paging operation had first priority) and software measurements were within 5% of hardware readings.

Hybrid monitors are combinations of hardware and software monitors. Software routines are used to move data of interest into special monitoring registers. The hardware acquisition unit then records the information from these registers. Ideally, this approach could reduce the individual disadvantages of separate hardware and software monitors. A significant disadvantage of hybrid monitors is that they require special hardware in the computer for the monitoring registers, and this requirement is impractical unless included in the original system architecture. Hybrid monitors are therefore very rarely used, although an interesting realization is discussed in [SEBA74].

In summary, hardware monitors have an advantage over software monitors in that they do not interfere or affect system performance by using resources and introduc-

ing overhead. They can record very high event rates, and times can be measured very precisely. The disadvantage of hardware monitors is that although they can reveal *what* is happening, they usually cannot reveal *why* an event is happening. In other words, they can record only physical states of the system, such as register contents; but they cannot measure logical information, e.g., which program caused an event to happen.

2. EXAMPLES OF SPECIFIC MEASUREMENT MONITORS AND SYSTEM ARCHITECTURES

This section will describe a few system dependent measurement monitors for some of the more popular families of computers. With the exception of the requirement to establish a probe point library for each type of computer, hardware monitors are generally system independent. All of the specific monitors to be presented will therefore be software monitors, and these monitors will be presented in terms of their capabilities for assisting in the modeling process. The presentation will be oriented toward the software monitors supplied by mainframe vendors. Other software monitors do exist, and many of these monitors collect data that, in varying degrees, can be used to support computer systems modeling. For the purpose of this tutorial article, however, the field is too extensive to discuss each individual monitor. The discussion for each computer will also highlight any architectural properties that affect modeling, as well as any particular considerations that affect hardware monitoring.

The IBM, Honeywell, and Univac monitors that will be discussed are a great improvement over those available in 1975, but prospective analysts should not be deceived into thinking that they are representative in all cases. And even these monitors could be improved in order for the analyst to take full advantage of some of the recent developments in analytical QNMs, e.g., the capability of specifying classes of jobs. In most other cases, commercially available software monitors do not measure all of the QNM input parameters. In these cases the

analyst will have to write the necessary software measurement routines. (Similarly, the necessary probe points that would enable hardware monitors to obtain directly certain model parameters may have not been researched and documented for some computers.) It is felt that this situation is not a reflection on the technical difficulty in achieving the necessary capabilities, but rather reflects the absence of such a requirement in the design specifications for the monitors.

The absence of convenient means and techniques for obtaining QNM input parameters for all computers can perhaps be explained as follows. First, measurement devices have historically been provided for the data processing center to determine system bottlenecks and to balance CPU and I/O processor utilizations. Typical measurements in this environment include "CPU waiting and only channel 1 busy" and "CPU busy and no channel busy". As will be shown in Section 5, there is absolutely no resemblance between these measurements and what is needed as input parameters for an analytical model. Second, because there has been such a broad spectrum of computer models, it is perhaps understandable that modeling requirements have not been seriously considered by designers of monitors. There now appears to be a perceptible trend toward the queueing network class of analytical models, and it is strongly recommended that future measurement devices should include the capability for measuring the required input parameters and output performance measures for this class of models.

IBM 370 Series

IBM's introduction of the Resource Measurement Facility (RMF) software in 1976 for computers with the OS/VS2 MVS operating system considerably improved the capability for modeling IBM computers. In general, RMF provides all of the necessary statistics to enable the analyst to compute the input parameters for QNMs using the methods of Section 4, Measurement Procedures. Information collected by RMF is presented in formatted reports, and in some instances can be plotted for a graphic overview.

Information that can be graphically plotted which is of interest to QNM analysts includes the number of batch and TSO users, CPU utilization, channel utilization for each channel, channel activity rate per second, mean channel service time, device activity rate, and mean device service rate. The graphical plotting feature permits the analyst to visualize rapidly the dynamics of system behavior for the particular monitoring session. For example, during a typical monitoring session of one hour the analyst might want to plot the above activities at five minute intervals and obtain a time history of demand and usage.

RMF gathers data by both event trace and by sampling methods. The user can specify when to obtain a trace, and can add fields to the list of traceable information. In the sampling method, certain data can be collected by RMF from the appropriate system counters at the beginning of the reporting interval and again at the end of the session. Since the counters are increased continually as events take place, RMF can determine an exact count for a process by calculating the difference between the counters at the start and completion of the interval. In contrast, other data counts (such as the number of jobs in main memory) fluctuate over an interval because of system requirements. This data is acquired by periodically sampling the counters at user specified intervals during the reporting period and then calculating the minimum, mean, and maximum value of the samples taken.

Another useful item provided by RMF is the distribution of CPU and I/O queue lengths. This data might prove useful during validation, since the model provides mean queue lengths by device. Most validations have been concerned with matching model device utilizations and throughputs with measured values. It would also be helpful to compare model and measured device mean queue lengths.

One feature not currently in RMF that is required to support the models of [CHAN75b, SAUE75, and SHUM77] is the capability to compute the standard deviation of device service times. However, even without this capability, RMF is a consid-

erable improvement over what was commercially available for the modeling analyst before 1976.

Other IBM-supplied software of interest to the analyst includes the Systems Management Facility (SMF) and the Generalized Trace Facility (GTF). SMF is an event trace monitor that records data at times of job step termination and job completion. Designed primarily as an accounting package, it records resource usage data for application programs, such as CPU time used, number of page faults, and the number of I/O operations. SMF does not record operating system statistics; therefore it is not suitable, in general, as a precise measurement monitor for the queueing analyst. SMF can, however, provide useful supplementary information if the analyst wishes to use the model of [BASK75 and CHAN75a] with timesharing and batch classes of jobs. Although RMF provides a count of the number of TSO and batch jobs, it does not allocate device utilizations and visit counts by class for these classes of jobs. Since SMF collects device utilizations and visit count data for both timesharing and batch application programs, it can be used to approximate operating system statistics for these classes. Examples are given in Section 5.

GTF is an event trace monitor for certain operating systems events, e.g., supervisor calls, I/O activity, paging behavior, and memory allocation. Information is recorded with time stamps, task identifiers, and status register contents. GTF can record an extensive amount of data, and, as a consequence, can impose a substantial amount of overhead. Benchmark experiments in [KIEN77] showed that GTF can increase the CPU utilization by a factor of two, not all of which is identified as such.

At this point it is instructional to discuss certain features of the IBM architecture of interest to the computer system analyst. IBM 370 series computer systems include either 2314 disks, which are connected to the channel during rotational latency and data transfer, or the newer 3330, 3340, and 3350 disks with rotational position sensing, which are connected to the channel only during data transfer. Validations can be

quite easily accomplished for systems with IBM 2314 disks, using the procedures of Section 4. Validations may be more difficult for the systems with the 3330s, and the operation of the 3330 class of disk will now be described in detail.

This type of disk, (which also includes the INTEL 7330), is divided into radial sectors. An initial seek command positions a read/write (R/W) head on the desired track. The channel then provides to the disk the sector number of the data record to be accessed. Unlike earlier I/O architectures (such as that associated with IBM 2314 disks), the channel can now be released from this disk to serve some other disk. The disk starts an offline search to find the requested sector. When the sector draws near the R/W head (perhaps two or three sectors in advance of the desired record), the disk sends a ready signal to the channel. Unless the channel acknowledges this message during the remaining time it takes the sector (record) to reach the R/W head, the sector passes under the R/W head without connecting, and data transfer will not occur until at least another full rotation. The device ready signal is issued each time the sector approaches the R/W head, and the action is repeated until the channel acknowledges and is connected.

Rafii [RAFI76] reported hardware measurements on an IBM 3330 spooling disk at Stanford University during different hours of the day. The measurement results show that during peak system usage periods (up to approximately 50% of the available time), the initial device ready signal was not received in time by the channel because of the heavy loading. During these time periods, it was not uncommon to experience delays of more than one rotational period. To model computer systems with this type of peripheral devices accurately, we should measure the percentage of delayed rotations. This subject will be discussed in more detail in Section 4.

Probe point libraries for IBM computers are reasonably complete. Comparisons of utilizations from hardware monitors and RMF indicated readings within 5%. Validations of QNMs on IBM computers have

been reported in [BHAN74, GIAM76, KIEN77, LIPS77, MOOR71, PRIC76, and ROSE76].

Honeywell 6000 Series

The Generalized Monitor Facility (GMF) has recently been developed for Honeywell 6000 series computers by the Department of Defense for computer performance monitoring of these systems in the World Wide Military Command and Control System. GMF is an event trace type of software measurement device and consists of six individual monitors for main memory, mass storage, CPU, tape drive, channel, and communications activity. Any combination of these six monitors can be in operation simultaneously; however, if all six are in operation simultaneously the amount of overhead imposed by the event trace approaches 10%.

The Memory Utilization Monitor, the Mass Storage Monitor, and the CPU Monitor measure considerable data of interest to the analyst. They provide a total of 58 reports, including the number of jobs in main memory, CPU utilizations, CPU service times, and the number of jobs in the CPU queue. Each report consists of a histogram and tabulated values of individual and cumulative probabilities of the events. The CPU service time activity is often of interest to the queueing analyst, since it will indicate how closely the actual CPU service time distribution approaches the assumptions of the model. This particular report gives a plot of the percent probability of occurrence for the service times in 1.5 millisecond intervals, as well as the individual and cumulative probabilities of each interval.

The Channel Monitor provides histogram and probability data for each I/O channel for service times, queue lengths, and waiting times. There are also reports for each device showing queue lengths and waiting times. The Tape Monitor is similar, with reports for the number of drives in use, waiting time for each tape drive, and an activity report by job.

The Communication Monitor provides information relative to terminal and remote

job entry activity. A device summary report shows mean and standard deviation values for session length, input/output character length, think times, and terminal response times for each device. Of particular interest are reports showing histograms and probability data for terminal response times, think times, and session lengths.

There has been a moderate amount of hardware monitoring of Honeywell 6000 computers, and a reasonably complete library of probe points exists. The CPU and channel utilization probe points have been validated and documented. Comparisons of results from hardware monitors and GMF have been conducted and show agreement within 5% [WENK77].

The most common I/O disks for Honeywell 6000 machines are the model 191 and model 451. Both types of disks remain connected to the channel during data transfer and rotational latency. (They are similar in this respect to the IBM 2314s.) For a validation and study of QNMs on a Honeywell 6000 series computer, see [DIET77].

Univac 1110 Series

The Univac-supplied software monitor is called the Software Instrumentation Package (SIP). It is an event-driven monitor that can be incorporated into the system at five different levels, as specified by system generation parameters. The first level of statistics consists primarily of processor utilizations and I/O summary information. Overhead introduced by this level is relatively low, e.g., of the order of 5 to 10%. Information from these reports is normally used for optimizing peripheral and mass storage configurations. These reports can also form the basis for determining QNM input parameter values.

The second level of statistics furnish the same information as the first level, plus additional data on distributions of memory utilizations and program sizes. Naturally, this level introduces more CPU, channel, and I/O unit overhead. The remaining three levels provide considerable detail, and are primarily useful for personnel working in areas of operating system development and modification.

Only the first level SIP statistics will be discussed here. The processor report provides CPU utilization data, plus a table listing the number and source of CPU interrupts. Another report shows the mean number of jobs executing in main memory for both batch and timesharing classes. Channel utilization, number of access control words, and number of data words transferred are reported for each channel in another summary. There is also an I/O trace monitor available for the 1110 that will record information on every I/O transfer.

There does not appear to be any available data comparing the accuracy of SIP with appropriate hardware measurements. CPU and channel utilization probe points have been documented.

Two types of disks are commonly found on the 1110 series. The 8414 disks remain connected to the channel during rotational latency and data transfer. Model 8433 disks are similar to IBM 3330s and disconnect from the channel during rotational latency. For examples of QNM validations and studies on Univac 1110s, see [BUZE77 and KRZE77]. (See also [HUGH73] for a validation on a Univac 1108.)

DEC 20 and 11/70 Computers

The analyst will find the modeling process for these computers more difficult, at least initially, than for systems previously discussed because of the absence of readily available instrumentation capabilities. There is no dedicated measurement software provided by the manufacturer for the 11/70 computer, and the author is not aware of any available measurement software suitable for supporting QNMs. Some of the information required for modeling is available in the operating system, but there are no software routines to read the data, and there are no data reduction routines for processing the data. As a result, modeling projects for the 11/70 computers should budget in advance a sufficient amount of time and money to obtain the required capabilities. There seems to have been no hardware monitoring of 11/70 computers, and consequently probe point libraries are not available.

The situation is slightly better for the

DEC 20 series. Software is provided that measures the number of jobs executing in main memory and the CPU utilization. Unfortunately, there are no provisions for obtaining channel utilizations or for obtaining the number of I/O operations by channel. The information seems to be available in the system, but there is no software for reading and processing the data. Again, the analyst should plan a certain amount of time and money to develop and test the measurement software. It appears that there has been little hardware monitoring of DEC 20 computers, and probe point libraries are not available from the vendor or from measurement groups.

CDC 6600 Series

CDC 6600s generally vary widely from system to system. The SCOPE operating system is initially provided by the vendor, but invariably users seem to tailor it to suit their own particular needs. Thus a commercially available monitor would have to be modified to some degree for each particular configuration, and possibly for this reason there appears to be no commercially available system software measurement monitor for CDC 6600 computers. There are some program monitors available through the user community (e.g., from NASA's Langley Research Center, and Mobil Oil Co., Dallas). Without a commercially available system software monitor, users have been forced to develop measurement routines for their own individual system and measurement objectives. Developing system software monitors for CDC 6600s is generally within the capabilities of good system programmers, and this approach has been successful in most cases.

There seems to have been very little hardware monitoring of CDC 6600s. Several years ago the speeds of hardware monitors were no faster than the electronic speeds of the 6600, and consequently, resolution of the monitor was often not sufficient to obtain the necessary readings. As a result, a reasonably complete library of validated and documented probe points was not developed. Today, even though electronic speeds of hardware monitors have improved by an order of magnitude,

using hardware monitors on CDC 6600s is still not common. A possible explanation is the investment cost of developing the library, and the fact that there is not a sizeable market to share the costs of the development. Validated probe points do exist for obtaining measurements of CPU and channel utilizations.

The most common disk for this series of computers is the CDC 844, which remains connected to the channel during both rotational latency and data transfer. Good results in validation have been obtained using the measurement procedure of Section 4. Examples of validations of a QNM for CDC 6600s are shown in [BASK72 and SCHW70].

3. DESCRIPTION OF A QUEUEING NETWORK MODEL

We illustrate measurement procedures in general by discussing a measurement procedure for a specific class of QNMs. This model will be referred to as the *baseline* model. It does not have the full generality of the model in [BASK75]; for example, there are no load-dependent devices or class changes. The particular features of the model were selected to make it typical of the class of QNMs that have been validated for many computers with excellent results. We describe the model here, and the measurement procedures in the next section.

The baseline model is a central server model with multiple classes of jobs (Figure 1). It is assumed that an integer number of jobs of each class traverse the closed network consisting of the central server (CPU) and the peripheral devices (I/O devices). Because we are restricting ourselves to a central server model, a job in the model alternately receives service from the CPU and one of the I/O devices. Most modern I/O architectures consist of channels, controllers, and actual I/O units (e.g., disks, drums, tapes), with the channels and I/O units providing service delays. For this model the analyst must select whether the I/O device should be either: 1) a "composite" channel with its associated I/O units; or 2) an individual I/O unit, each with an implicit, "virtual" channel.

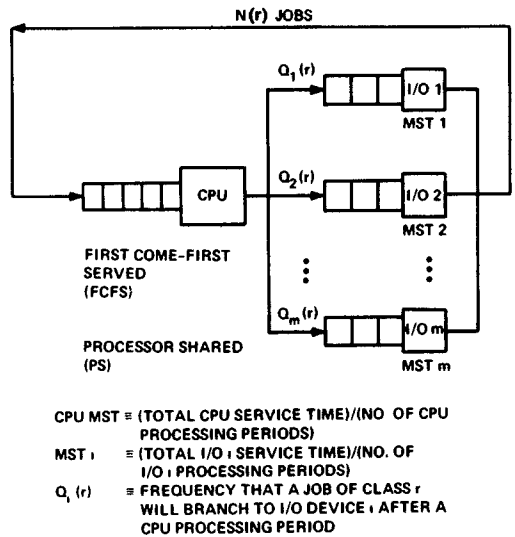


FIGURE 1. Closed queueing network model.

Queueing disciplines and service time distributions must be specified for each device (we are using the stochastic approach to QNMs). Certain of the decisions are motivated by the way computer systems actually operate. Other decisions are forced by the particular solution technique.

In actual systems, I/O devices are generally scheduled in a First-Come-First-Served (FCFS) discipline; therefore this choice was selected as the model I/O queueing discipline. Given this choice, however, the I/O service time distribution is restricted: At the *i*th I/O device, there is an exponential service time distribution with mean *I/O i* MST, which is the same for all classes. Different I/O devices may have different exponential distributions, however.

Two common forms of CPU scheduling in actual systems are FCFS and round-robin. FCFS scheduling at the CPU has the same modeling restrictions as that of an I/O device. Round-robin scheduling cannot be modeled exactly, but a close form of it, processor sharing (PS), can be; see Section 4. PS scheduling at the CPU is not as restrictive from a modeling viewpoint as FCFS. The CPU service time distribution may be class dependent. Also, service time distributions more general than exponential are allowed; these distributions are known to yield utilizations and queue length dis-

tributions identical to those obtained for exponential distributions [BASK75].

The routing behavior of jobs may be class dependent. After completing service at the CPU, a job of class r selects I/O device i according to a routing frequency $Q_i(r)$, associated with that device and (possibly) that class. Figure 1 does not contain a feedback loop from the CPU back to itself [DENN78, see p. 225, in this issue]; system throughput can still be obtained by knowing the CPU throughput and the number of CPU visits per job.

Thus, the required input parameters for this baseline model are:

- 1) Number of classes of jobs (R).
- 2) CPU queueing discipline (FCFS or PS).
- 3) Degree of multiprogramming ($N(r)$), by class.
- 4) I/O device routing frequencies ($Q_i(r)$), by device and by class.
- 5) CPU mean service time ($CPU\ MST(r)$), by class if PS.
- 6) I/O device i mean service time ($I/O\ i\ MST$), by device.

The calculated output performance measures from the model are:

- 1) Utilization, by device (and by class for a PS CPU).
- 2) Throughput, by device (and by class for a PS CPU).
- 3) Mean queue length, by device and by class.
- 4) Mean waiting time in the queue, by device and by class.

The model is evaluated using the formulae in [BASK75], with the normalization constant (for closed networks) determined by the method in [CHAN75a].

The baseline model is relatively simple; it reduces to Buzen's central server model [BUZE71, 73] when there is a single class of jobs. The baseline model has pedagogical advantages; once the measurement requirements and procedures are understood for the baseline case, it is relatively easy to extend them to the more complex cases of 1) QNMs for closed networks that permit non-exponential service time distributions (e.g., using the approximate solution techniques in [CHAN75b, SAUE75, SHUM77]), and 2) QNMs for mixed (open and closed)

networks that permit arrivals and departures of jobs, with the normalization constant determined by the method in [REIS75].

4. MEASUREMENT PROCEDURES FOR OBTAINING QUEUEING NETWORK MODEL INPUT PARAMETERS

This section will present a set of procedures for obtaining QNM input parameters. This set of procedures should be thought of as a structured yet flexible set of guidelines, since the particulars will vary somewhat from computer to computer. Also, this set should be evolutionary to reflect both the refinement of analytical models of computer systems and the development of instrumentation that will more directly support these models. The measurement procedure discussed in this paper has been validated and has resulted in useful applications of the model. (Some examples will be shown in the next sections.)

Number of Classes of Jobs

This first parameter is specified according to the objectives of the analysis, the job environment, and whether one has the instrumentation capability to collect separate statistics for each class. Assuming that the necessary instrumentation is available, an analyst might want to model a computer system using two classes of jobs, e.g., time-sharing and batch. Another example might be to use different classes to model job priority classes. In general, the difficulty in using classes of jobs is that measurement monitors may not provide the necessary information that will enable the analyst to compute statistics by class. Thus the analyst would, in these cases, have to estimate or approximate class statistics using the best available information. A procedure for estimating class statistics for timesharing and batch jobs when modeling IBM computer systems is described in detail in [ROSE75]. See also [KRZE77] for an example with Univac systems.

CPU Queueing Discipline

The CPU queueing discipline may be specified in the baseline model as either First-

Come-First-Served (FCFS) or Processor Sharing (PS). PS is a form of "round-robin" scheduling, in which time-slicing is used to prevent a task from holding a processor unduly long. In effect, the scheduler loads an interval timer clock with a time quantum. If the task has not released the processor by the expiration of the quantum, the clock will generate an interrupt that will cause the task to be preempted and a new task assigned. In PS, the length of the quantum approaches zero, and each of n jobs will receive service at $(1/n)$ th the rate that a single job at the CPU would receive. For further discussion of the PS discipline, see [COFF73 and LAMP68].

The CPU queueing discipline is specified for the model, and ideally is determined from a knowledge of the workload and computer operating characteristics. As an example, for a CDC 6600 at the University of Texas (Austin), the mean time between CPU interrupts is about 30 milliseconds (ms). The CPU quantum is set for 16 ms.; therefore, most jobs will be preempted before experiencing a CPU interrupt. Consequently, PS should be a better approximation for this operation than FCFS. [BASK72] shows that PS does yield good results in a validation on this system.

As another example, under the IBM VS2 operating system the CPU quantum is set for 500 ms. For a workload with a *CPU MST* of approximately 20 ms., one would expect that a job would complete its full CPU processing before a timer interrupt occurs. This operation should be more accurately modeled by FCFS than by PS. The FCFS CPU discipline was in fact more accurate for the experiments on IBM computers in [ROSE76], in which the *CPU MST* ranged from 10 to 35 ms.

Degree of Multiprogramming

The number of jobs in the closed QNM of Figure 1 will, in general, relate to the number of jobs executing in the main memory. The model assumes that a fixed integer number $N(r)$ of jobs of class r are in the closed queueing network. These jobs may be in one of the following states: 1) in the CPU queue, 2) receiving service from the

CPU, 3) in an I/O device queue, or 4) receiving service from an I/O device.

To determine $N(r)$, one can normally measure (using a software monitor) the time-averaged number of jobs actually in execution in main memory during the period of the test. This information is generally readily available in the operating system, and can be retrieved using either sampling or event trace techniques. When using a sampling software monitor, it is often sufficient to measure the number of jobs in execution at 30-second intervals during the period of the test and compute the time-averaged $N(r)$ from this data. Due to the dynamic behavior of programs, the measured degree of multiprogramming will usually not be an integer. Since the computational algorithms assume integer values of $N(r)$, the analyst may wish to interpolate between model output values for the two closest integer values of $N(r)$. In other instances, if there are many parameters to be investigated, the analyst may want to round off $N(r)$ to the nearest integer values.

I/O Device Routing Frequencies

The model assumes that a job receives alternate processing from the CPU and an I/O device. The routing frequency from the CPU to an I/O device is defined as the number of I/O operations (reads, writes, or paging operations) for that device divided by the total number of I/O operations. There are two possible measurement techniques to obtain this parameter. One approach is to use a software monitor to count the number of times that the I/O data transfer macro was successfully executed. Another approach is to use a hardware monitor to count the number of times that a channel or a peripheral device on a channel transferred or received a block of data.

An I/O device in the model may represent either: 1) a channel and its associated peripherals collectively, or 2) each individual peripheral unit (without the channel included explicitly). For the baseline model, the analyst may select which of these two representations to use based on such factors as the objectives of the analysis and the capabilities of the available measurement monitors.

The model permits class-dependent routing frequencies for R classes of jobs. Referring to Figure 1 and defining $I/O\ OP(r)$, as the number of I/O operations for device i for class r jobs, the routing frequency from the CPU to device i for class r jobs is:

$$Q_i(r) = \frac{I/O\ OP(r)_i}{\sum_{i=1}^m I/O\ OP(r)} \quad r = 1, \dots, R \quad (1)$$

CPU Mean Service Time

Conceptually, this parameter can be obtained by dividing the total CPU service time by the number of CPU processing periods. Since existing measurement monitors (hardware or software) usually do not accumulate CPU processing times, it is necessary to derive the total service time for the CPU by measuring the CPU utilization ($CPU\ UTIL$) and multiplying it by the length of the test period (T). (CPU utilization is defined as the fraction of time that the CPU is in either the problem state or the supervisor state.) Dividing CPU service time by the number of CPU processing cycles will yield the CPU Mean Service Time ($CPU\ MST$). The number of CPU processing cycles in this model is equal to the total number of I/O processing cycles. Assuming a FCFS scheduling discipline, the $CPU\ MST$ is class independent and may be determined by:

$$CPU\ MST = \frac{(CPU\ UTIL)(T)}{\sum_{i=1}^m I/O\ OP} \quad (2a)$$

For the case of a PS queueing discipline, the $CPU\ MST$ s may be class dependent. Assuming that $f(r)$ is the fraction of total CPU utilization used by class r jobs:

$$CPU\ MST(r) = \frac{f(r)(CPU\ UTIL)(T)}{\sum_{i=1}^m I/O\ OP_i(r)} \quad r = 1, \dots, R \quad (2b)$$

In general, measurement monitors do not collect information on the amount of supervisor (operating system) overhead required for each class of job. Until this capability is available, one way of handling the overhead is to allocate it in Equation (2a) across all jobs. For the validations in [ROSE75] it was shown that using only problem state utili-

zation gave unsatisfactory results, and measurement data did not provide statistics to permit assignment of an "operating system" job to account for the time spent in the supervisor state. Similarly in equation (2b), due to the limitations of current measurement monitors, the overhead is allocated across the classes. It is shown in [ROSE75] how $f(r)$ can be approximated for IBM machines for classes of timesharing and batch jobs.

I/O Device Mean Service Time

It is, in actual practice, very difficult with current analytical techniques and measurement monitors to model and measure the I/O behavior of modern computer systems exactly. From the analytical aspect, it is extremely difficult to model the parallelism of channel operation when several devices are in various modes of operation, such as seek, rotational latency, data transfer, waiting for a control unit, and waiting for a channel. Analytical techniques for modeling overlap are not currently available, and this problem will probably require much research and effort in the future. (See [CHAN78] p. 281, in this issue.)

Modern I/O architecture can also cause problems from the measurement point of view. To devise a measurement procedure for obtaining this input parameter, one should consider both the operation of the computer system and the assumptions of the model. The service time for a I/O device is equal to the seek time (if the device is a disk) plus the time of rotational latency plus the time of data transfer. Based on these assumptions of the model, the $I/O\ MST$ should be expected to include all of these components. Define $MTDT$ to be the mean time for data transfer, $MTRL$ to be the mean time for rotational latency, and $MTSK$ to be the mean time for a seek. Then:

$$I/O\ MST = MTDT + MTRL + MTSK \quad (3)$$

Equation (3) is the basis for the measurement procedures to obtain the I/O device input parameters. These procedures must take into account the two alternatives for modeling the I/O device, i.e., whether the device is a composite channel and peripheral structure (Case 1) or an individual pe-

ripheral device with a "virtual" channel (Case 2).

Nearly all hardware monitors and some software monitors have the capability of measuring individual disk seek times (Case 2). For Case 1, *MTSK* can be obtained by averaging the individual disk *MTSKs* for that channel.

The following discussion will be primarily applicable for Case 1, and it will develop a measurement procedure that, in effect, averages the individual peripheral units' *MTDTs* and *MTRLs* for a particular channel. (This procedure is often more convenient to use with current monitors than having to measure the individual *MTDTs* and *MTRLs*, and then to compute the average.) Certain of the principles will also prove useful for Case 2.

Before developing the procedure, it is helpful to recall the discussions in Section 2 concerning the various I/O architectures. A channel is always required for the transmission of I/O data, but some I/O architectures permit the channel to be released during rotational latency or seek. For example, the IBM 2314, CDC 844, and Honeywell 451 disks are connected to a channel during data transfer and rotational latency. The more recent IBM 3330 class of disks and Univac 8433 disks use rotational position sensing (RPS) and are connected to a channel only during data transfer.

Nearly all monitors provide the capability for measuring channel utilization. In a manner analogous to deriving *CPU MST*, the mean service time for channel *i* (*CH i MST*) will be:

$$CH\ i\ MST = \frac{(CH\ i\ UTIL)\ (T)}{I/O\ OP_i} \quad (4)$$

For the case of disks without RPS, measured channel utilization will include the service times for rotational latency and data transfer. Then:

$$CH\ i\ MST = MTDT + MTRL \quad (5a)$$

Thus Equation (4) and the procedure for obtaining *MTSK* provide all of the information needed in Equation (3) for disks without RPS.

For the case of disks with RPS:

$$CH\ i\ MST = MTDT \quad (5b)$$

For a large sample size, one would expect

that on the average the *MTRL* will be approximately one-half of the full rotational period. For example, the rotational period of IBM 3330 disks is 16.7 ms. A first approximation for *MTRL* would be 8.35 ms. However, this approximation does not take into account missed rotations, as discussed in Section 2. If one could measure or estimate the fraction of missed rotations, the first order approximation could be improved. Suppose that for an IBM 3330 system there were 10% missed rotations. Then a better approximation for *MTRL* would be $(8.35) (1.1) = 9.2$ ms. This procedure, in conjunction with Equation (4) and measured *MTSK*, completes the information needed in Equation (3) for disks with RPS.

For Case 2, Equation (3) clearly indicates the required measurements for *MTDT* and *MTRL* for the individual devices. The above procedures for deriving *MTRLs* are useful here also. Most monitors provide information that can be used to derive the individual device *MTDTs*. Because RPS operation can pose problems in modeling, the reader is also referred to [WILH77 and ZAHO77].

Optional Parameters

The baseline QNM of [BASK75 and CHAN75a] assumes that the service times of FCFS processors are exponentially distributed. The models of [CHAN75b, CHAN78, SAUE75, and SHUM77] relax this restriction and permit approximate solutions for nonexponential service time distributions. (The Sauer-Chandy model also permits CPU priority disciplines.) The input parameters for these models are essentially the same as for the baseline model, except that the coefficient of variation of service time is required for the CPU and I/O devices. (The coefficient of variation for a distribution is defined as the ratio of the standard deviation to the mean.) Methods for determining the mean of the service times have already been discussed. Hence the additional measurement requirement imposed by these models is to obtain the standard deviation of service time distributions. Some, but not all, measurement monitors available today are capable of obtaining the standard deviation. Thus if analysts wish to use these more generalized

models, they must insure that they can obtain a monitor with this capability.

For the mixed (open and closed) queueing network model of [REIS75], the analyst would need to measure arrival rates and departure routing frequencies in addition to the parameters for the baseline model. Departure routing frequencies can be obtained in a manner analogous to I/O routing frequencies. The exact technique for obtaining arrival rates will vary depending upon the particular computer system, but a possible approach would surely be an event-driven software probe at the system job queue.

5. MODEL VALIDATIONS

The previous section presented a suggested set of procedures for obtaining input parameter values for a representative QNM. The output performance measures of the model are: 1) device (CPU and I/O) utilizations, 2) device throughputs, 3) device mean queue lengths (MQL), and 4) device mean waiting times (MWT).

The objective of the validation process is to compare calculated performance values with corresponding measured values and to resolve any discrepancies. With the currently available analytical and measurement tools, model validation is still a combination of art and science. The analyst should keep in mind, however, that QNMs have produced good results provided that the input data was both accurate and consistent with the assumptions of the model.

Since virtually all measurement monitors provide device utilizations, the first step in the validation process is to compare model and measured utilization data. The next step is to compare throughput values. Some software monitors provide throughput directly; in other cases it can be calculated easily. For example, CPU throughput is simply the number of service requests processed by the CPU divided by the monitoring interval (T). Other device throughputs can be calculated similarly, or by using the Job Flow Balance equations in [DENN78].

Hardware monitors record the time that a device is in a certain state, or the contents of registers. Neither throughput nor MQL is normally obtained by hardware monitors.

Some software monitors do measure data to compute MQL, in which case the model MQL can be validated. The measured MQL can be used in Little's law [DENN78] to calculate MWT, thereby validating this output value.

During the early validation trials it is strongly recommended that a benchmark, or fixed jobstream, be used to ensure repeatability of the measured data and to establish the basis for controlled experiments. The experiments described in this paper were conducted during dedicated time (with no other users of the system) and used a benchmark obtained from the Federal Computer Performance Evaluation and Simulation Center (FEDSIM). This particular benchmark is capable of generating both batch and timesharing jobstreams. It executes 60 job steps for batch processing during a running time of approximately 20 minutes, and provides a mix of CPU bound, I/O bound, and balanced workloads. For timesharing operation, it executes a circular list of commands for such typical timesharing tasks as editing, allocating, and linking.

The validation to be described was made on an IBM 370 Model 155-2, with virtual storage operating system VS2, HASP, and the Time Sharing Option (TSO). The system configuration consisted of one CPU, 1.5M bytes of main storage, and four channels.

For this validation the 370/155-2 system was instrumented with a hardware monitor and with a software monitor similar to RMF. This particular hardware monitor permitted a maximum of 20 probes in the computer system. As a result of this limitation, I/O devices were modeled using the approach of composite channel and peripheral devices. Hardware probes were then placed to measure CPU and channel utilizations, and disk seek times. The software monitor was used to measure $I/O\ OP(r)$, and $N(r)$.

Referring to the block diagram in Figure 1, channel 1 was a byte multiplexer channel and provided a data path through a control unit to a printer and a card punch. Channel 2 was a block multiplexer channel and provided a path to four 3330 disks via a Disk Control Unit (DCU). The remaining two

channels were selector channels which connected to independent banks of 2314 disks via DCUs. There were eight 2314s connected to channel 3. Channel 4 had eight 2314s and eight timesharing terminals.

The objective of these experiments was to study system behavior rather than to examine the detailed operation of a particular subsystem. This objective did not suggest the use of a detailed or an embedded paging model. Hence, the paging statistics were included as an integral part of system I/O statistics, e.g., the number of page-in/page-outs were included in the computation of the routing frequencies for the channel with the paging pack. Of course, this simplification may not be valid in all cases, but good results using this assumption were achieved for this validation.

For the first experiment, the system was configured without the IBM 3330 disks, i.e., channel 2 was not used. To simplify generation of the operating system, no timesharing terminals were used, and only the batch portion of the benchmark was needed.

The objective of this first experiment was to validate the model and the measurement procedures in Section 4 for a single class of jobs (batch). Model input parameter values were obtained using the set of procedures in Section 4. The time-averaged degree of multiprogramming was measured at 3.7, and model utilizations were solved for $N = 3$ and $N = 4$. Results are shown in Table I.

The next set of experiments on the 370/155-2 was to investigate the use of job classes of timesharing and batch. In order to use the model with multiple classes, the routing frequencies must be determined for each class. CPU MSTs may be partitioned by class if one uses the PS queueing discipline. Recall that channel and CPU MSTs

with a FCFS queueing discipline use statistics that are independent of class.

The software monitor obtained accurate counts of total I/O operations suitable for Equation (1), but it did not collect these statistics by class. SMF collects I/O statistics for classes of timesharing and batch for application programs, but not for the operating system tasks. One method of approximating $Q_i(r)$ is to modify Equation (1) to include a weighting factor $W_i(r)$, which is the relative fraction of I/O OPs for that channel resulting from TSO and batch applications programs:

$$Q_i(r) = W_i(r) \frac{I/O\ OP_i}{\sum_{i=1}^m I/O\ OP_i} \quad r = 1, \dots, R \quad (6)$$

Suppose that SMF recorded 4000 batch and 3000 TSO I/O OPs to the paging pack during concurrent batch and timesharing operation, and suppose that the software monitor recorded 9000 total I/O OPs to the paging pack. (These were typical statistics using this benchmark.) Then the number of TSO I/O OPs would be approximately $[3000/(3000 + 4000)]9,000$.

In other instances, using the above approach for obtaining the weighting factor $W_i(r)$ in Equation (6) will not be possible. An alternative method that is feasible when using a benchmark is to first collect I/O statistics when the system is executing just the batch component of the benchmark. Next, I/O statistics can be measured when the system is executing the timesharing component of the benchmark. The weighting factor for concurrent batch and timesharing operation is the relative fraction derived from these two sets of data. This particular approach for obtaining $W_i(r)$ was used for the system job queue, because SMF does not count any I/O OPs to that data set. For additional details concerning these estimation techniques, see [ROSE75].

SMF data was used to determine $f_i(r)$ in Equation (2b). CPU MSTs for timesharing and batch were 35 ms. and 11 ms., respectively. Recall from Section 4 that for these values of CPU MST (r), one would expect FCFS to be more accurate for IBM computers than PS. A comparison of both CPU queueing disciplines using these classes in the model is shown in Table II.

TABLE I. MODEL VERSUS MEASURED UTILIZATIONS FOR THE CASE OF IBM 2314 DISKS ON AN IBM 370/155-2

DEVICE	MODEL UTILIZATIONS $N = 3$	MEASURED UTILIZATIONS $N = 3.7$	MODEL UTILIZATIONS $N = 4$
CPU	0.648	0.696	0.733
CH 1	0.544	0.576	0.615
CH 3	0.372	0.400	0.421
CH 4	0.394	0.422	0.445

TABLE II. A COMPARISON OF MODEL AND MEASURED UTILIZATIONS FOR AN IBM 370/155-2 USING FCFS AND PS CPU DISCIPLINES

DEVICE	MODEL UTILIZATIONS CPU:FCFS	MEASURED UTILIZATIONS	MODEL UTILIZATIONS CPU:PS
CPU	0.890	0.893	0.865
CH 1	0.433	0.449	0.517
CH 2	0.209	0.211	0.203
CH 3	0.318	0.320	0.355
CH 4	0.476	0.478	0.518

6. PERFORMANCE PREDICTIONS

Queueing network models are a useful class of models to analyze computer systems, but more studies are needed to determine their ranges of applicability. There have been surprisingly few studies in which computer performance predictions from an analytical model have been compared with actual measurements. The concept of using an analytical model to predict performance after a reconfiguration, performing the reconfiguration, and comparing model and measured utilizations was the basic theme of the projects reported in [HUGH73 and ROSE77]. Their approach was to run a benchmark program on the baseline system and to obtain the model parameter values. After validation, new model input parameter values were estimated in order to predict CPU and I/O channel utilizations for the upgraded or reconfigured system. Lastly, the benchmark was rerun on the reconfigured system and measured utilizations were compared with predicted values from the model. Reconfigurations reported by [HUGH73] included reallocating files and increasing main memory on a Univac 1108 with EXEC OS. Reconfigurations reported by [ROSE77] included these same two experiments, plus a reconfiguration involving an increase in the number of channels for a system. These experiments were conducted on an IBM 370/155 with OS/MVT, 370/155-2 with OS/VS2, and 370/168-1 with OS/VS2. Only the experiment for increasing the number of channels on the 370/155-2 will be discussed here.

The model had been validated on the 370/155-2 with four channels as shown in Table II in the previous section. To predict

the performance of a six-channel configuration, the analyst must estimate new input parameter values for the model. The estimation procedure uses the equations of Section 4, in conjunction with a knowledge of the principles of operation of the system hardware and software.

For example, the first step was to estimate new routing frequencies for the model. The channel allocation algorithm operating in the 370/155-2 VS2 system seeks to distribute equally across the channels the I/O OPs from the batch portion of the benchmark. Channels 5 and 6 for the reconfigured system did not have TSO terminals. Using this information and measured data from the four-channel validation, a table of estimated number of I/O OPs by channel and by class can be constructed, based on the following invariant assumptions:

- 1) The number of batch and TSO I/O OPs remain the same.
- 2) The number of batch I/O OPs are equally distributed across the channels.

Using this table and Equation (1), new device routing frequencies can be estimated. This table, in conjunction with measured data from the validation, will also permit new channel MSTs to be estimated, using Equation (4). Detailed procedures are described in [ROSE75]. It was assumed that the CPU MST and degree of multiprogramming remained unchanged. Predicted utilizations for the six channel system were then computed using the model. Model predictions can be compared in Table III with data actually measured.

SUMMARY AND CONCLUSIONS

To encourage increased use of analytical models, this paper has presented a measurement procedure for obtaining model input parameter values and output performance measures that is oriented toward a popular class of QNMs for computer systems. The approach utilizes to the maximum extent possible the parameters that are commonly provided by current measurement monitors.

Effective computer performance evaluation requires a reasonable understanding of

TABLE III. PREDICTED VERSUS MEASURED UTILIZATIONS WITH TWO CLASSES OF JOBS DURING CONCURRENT TIMESHARING AND BATCH OPERATION FOR ADDING TWO CHANNELS TO AN IBM 370/155-2

DE- VICE	MODEL UTILIZA- TIONS	MEASURED UTILIZA- TIONS	RELATIVE DIFFER- ENCE (%)
	$N(TSO) = 1$ $N(BATCH) = 4$	$N(TSO) = 0.8$ $N(BATCH) = 3.8$	
CPU	0.929	0.914	1.6
CH 1	0.482	0.484	0.4
CH 2	0.522	0.510	2.4
CH 3	0.140	0.143	2.1
CH 4	0.311	0.315	1.3
CH 5	0.074	0.083	10.8
CH 6	0.074	0.071	4.2

the assumptions of the model and a familiarity with the basic operation of the measurement monitor. The reason that both modeling and measurement disciplines are involved is that when one is interested in applications of QNMs for computer systems, the first problem to be solved is to determine what measurable quantities for the particular computer system most nearly correspond to the parameters in the model. Correct solution of this problem is of crucial importance in the validation. The model validation process will require: 1) understanding the assumptions and fundamental properties of the model; 2) understanding the architecture and operating system for the particular computer system; 3) verifying that the proper quantities are being correctly measured by the monitor; and 4) comparing calculated performance values with corresponding measured data and resolving any discrepancies.

This paper has presented the two basic approaches to monitoring computer systems (event trace and sampling), and the three types of measurement monitors (hardware, software, and hybrid). Also included is a detailed discussion of current measurement capability (as it applies to QNMs) for IBM 370, Honeywell 6000, Univac 1110, DEC 20 and 11/70, and CDC 6000 computer systems.

Many current commercially available or vendor-supplied measurement monitors do not permit the proper QNM input parameter values to be obtained readily. This situation does not result from the technical

difficulty in achieving the necessary capability, but rather reflects the absence of such a requirement in the design specifications for the monitors. It is hoped that this paper will create the motivation necessary to have the required capabilities included in future measurement monitors.

With state-of-the-art analytical and measurement tools, model validation and performance prediction is a combination of art and science. The analyst should keep in mind, however, that QNMs have produced good results provided that the input data was accurate and consistent with the assumptions of the model.

ACKNOWLEDGMENTS

I would like to acknowledge the contributions to this paper provided by Jeff Buzen and Scott Graham.

REFERENCES

BASK72 BASKETT, F., AND GOMEZ, F. P. "Processor sharing in a central server queueing model of multiprogramming with applications," in *Proc. Sixth Annual Princeton Conf. Information Sciences and Systems*, 1972, Princeton Univ. Press, Princeton, N.J., pp. 598-603.

BASK75 BASKETT, F.; CHANDY, K. M., MUNTZ, R. R., AND PALACIOS-GOMEZ, F. "Open, closed and mixed networks of queues with different classes of customers," *J. ACM* 22, 2 (April 1975), 248-260.

BHAN74 BHANDIWAD, R. A.; AND WILLIAMS, A. C. "Queueing network models of computer systems," in *Proc Third Texas Conf. Computing Systems*, 1974 Roma.

BUZE71 BUZEN, J.P. "Queueing network models of multiprogramming," PhD Thesis, Div. Eng. and Applied Science, Harvard Univ., Cambridge, Mass., (NTIS AD 731 575), Aug. 1971.

BUZE73 BUZEN, J. P. "Computational algorithms for closed queueing networks with exponential servers," *Commun. ACM* 16, 9 (Sept. 1973), 527-531

BUZE75 BUZEN, J. P. "Cost effective analytical tools for computer performance evaluation," in *Proc. IEEE COMPCON*, 1975, IEEE, New York, pp. 293-296.

BUZE77 BUZEN, J. P.; DORL, M.; GISH, J.; AND PINKERTON, T. "Experience with the Amperif solid state drum subsystem: hardware reliability and performance evaluation," in *Proc. Spring Conf. USE, Inc., Univac Users' Group*, 1977.

CHAN75a CHANDY, K. M.; HERZOG, U.; AND WOO, L. "Parametric analysis of queueing network models," *IBM J. Res. Dev.* 19, 1 (Jan. 1975), 36-42.

CHAN75b CHANDY, K. M., HERZOG, U., AND WOO, L. "Approximate analysis of general queueing networks," *IBM J. Res. Dev.* 19, 1 (Jan. 1975), 43-49.

CHAN78 CHANDY, K. M.; AND SAUER, C.

- H. "Approximate methods for analyzing queueing network models of computing systems," *Comput Surv.* 10, 3 (Sept. 1978), 281-317
- COFF73 COFFMAN, E. G., JR.; AND DENNING, P. J. *Operating system theory*, Prentice-Hall, Englewood Cliffs, N. J., 1973.
- DENN78 DENNING, P. J.; AND BUZEN, J. P. "The operational analysis of queueing network models," *Comput. Surv.* 10, 3 (Sept. 1978), 225-261.
- DIET77 DIETHELM, M. A. "An empirical evaluation of analytical models for computer system performance prediction," in *Proc Third Int. Symp. Computer Performance Modeling, Measurement and Evaluation*, 1977, North-Holland Publ. Co., Amsterdam, The Netherlands, pp. 139-160.
- GIAM76 GIAMMO, T. "Validation of a computer performance model of the exponential queueing network family," in *Proc. Int. Symp. Computer Performance Modeling, Measurement and Evaluation*, 1976, ACM, New York, pp. 44-58
- HUGH73 HUGHES, P. H., AND MOE, G. "A structural approach to computer performance analysis," in *Proc 1973 AFIPS National Computer Conf.*, Vol. 42, AFIPS Press, Montvale, N. J., pp. 109-119.
- KRZE77 KRZESINSKI, A.; GERBER, S., AND TEUNISSEN, P. "A multiclass network model of a multiprogramming timesharing computer system," in *Proc IFIP Congress 1977*, North-Holland Publ. Co., Amsterdam, The Netherlands, pp. 481-486.
- KIEN77 KIENZLE, M. G. *Measurements of computer systems for queueing network models*, Tech. Rep. CSRG-86. Also MSc Thesis, Univ. Toronto, Toronto, Ontario, Canada, Oct. 1977.
- LAMP68 LAMPSON, B. W. "A scheduling philosophy for multiprocessing system," *Commun ACM* 11, 5, (May 1968), 347-359.
- LIPS77 LIPSKY, L.; AND CHURCH, J. D. "Applications of a queueing network model for a computer system," *Comput. Surv.* 9, 3 (Sept. 1977), 205-221.
- LUCA71 LUCAS, H. C., JR. "Performance evaluation and monitoring," *Comput. Surv.* 3, 3 (Sept. 1971), 79-91.
- MOOR71 MOORE, C. G. "Network models for large scale time-sharing systems," PhD Thesis, Univ. Michigan, Ann Arbor, April 1971.
- PRIC76 PRICE, T. G. "A comparison of queueing network models and measurements of a multiprogrammed computer system," *SIGMETRICS Perform. Evaluat. Rev.* 5, 4 (1976), 39-62.
- RAFI76 RAFIL, A. "A study of the performance of RPS," *SIGMETRICS Perform Eval Rev* 5, 4 (1976), 21-38.
- REIS74 REISER, M.; AND KOBAYASHI, H. "The effects of service time distributions on system performance," in *Proc. IFIP Congress 1974*, North-Holland Publ. Co., Amsterdam, The Netherlands, pp. 230-234.
- REIS75 REISER, M.; AND KOBAYASHI, H. "Queueing networks with multiple closed chains: theory and computational algorithms," *IBM J. Res. Dev.* 19, 3 (May 1975), 283-294.
- ROSE75 ROSE, C. A. "Measurement and analysis for computer performance evaluation," ScD Dissertation, George Washington Univ., Washington, D. C., July 1975.
- ROSE76 ROSE, C. A. "Validation of a queueing model with classes of customers," in *Proc. Int. Symp. Computer Performance Modeling, Measurement, and Evaluation*, 1976, ACM, New York, pp. 318-325.
- ROSE77 ROSE, C. A. "A calibration-prediction technique for estimating computer performance," in *Proc. 1977 AFIPS National Computer Conf.*, Vol. 46, AFIPS Press, Montvale, N.J., pp. 813-818.
- SAUE75 SAUER, C., AND CHANDY, K. M. "Approximate analysis of central server models," *IBM J. Res. Dev.* 19, 3 (May 1975), 301-313.
- SCHW70 SCHWETMAN, H. D. "A study of resource utilization and performance evaluation of large-scale computer systems," PhD Thesis, Univ. Texas, Austin, 1970.
- SEBA74 SEBASTIAN, P. R. "Hybrid events monitoring instrument," in *Proc. 1974 SIGMETRICS Symp.*, ACM, New York, pp. 127-139.
- SEKI72 SEKINO, A. "Performance evaluation of multiprogrammed time-shared computer systems," PhD Thesis. Also MIT Project MAC Report MAC-TR-103, MIT, Cambridge, Mass., Sept 1972.
- SVOB76 SVOBODOVA, L. *Computer performance measurement and evaluation methods: analysis and applications*, Elsevier North-Holland, Inc., New York, 1976
- SHUM77 SHUM, A. W.-C.; AND BUZEN, J. P. "The EPF technique: a method for obtaining approximate solutions to closed queueing networks with general service times," in *Proc. Int. Symp. Computer Performance Modeling, Measurement, and Evaluation*, 1977, North-Holland Publ. Co., Amsterdam, The Netherlands
- WENK77 WENKER, W. J. Command Control Technical Center, Defense Communications Agency. Private communication, Nov 1977.
- WILH77 WILHELM, N. C. "A general model for the performance of disk systems," *J ACM* 24, 1 (Jan. 1977), 14-31.
- ZAHO77 ZAHORJAN, J. "A queueing model of an RPS disk system," in Tech. Rep. CSRG-83, Univ Toronto, Toronto, Ontario, Canada, July 1977

RECEIVED OCTOBER 12, 1977, FINAL REVISION ACCEPTED JUNE 6, 1978.