The design of the storage component is essential to the achieving of a good overall cost-performance balance in a computing system.

A method is presented for quickly assessing many of the technological and structural possibilities that exist today for designing storage hierarchies.

The evaluation is based on a cycling queuing model of the computer system and its programming environment, which are taken into account by miss ratio curves.

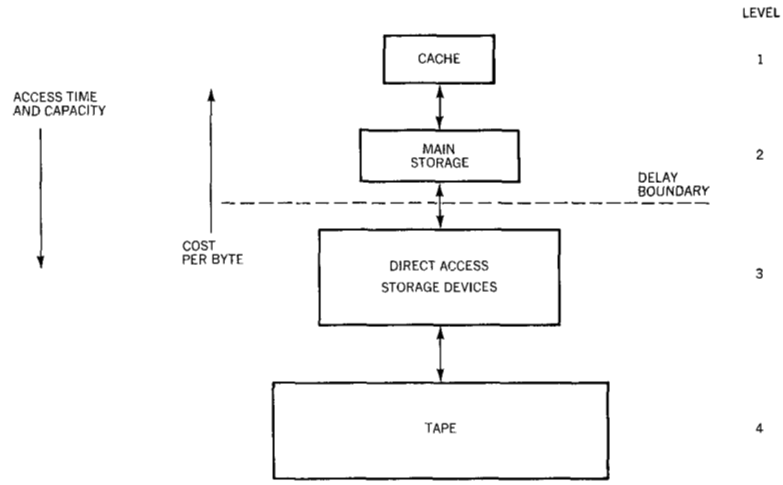
A model for the evaluation of storage hierarchies

by J. Gecsei, and J. A. Lukes

The success of a computing system depends not only on its technical qualities, but also—to a large extent—on its ability to match the cost, performance, and functional requirements of users. As a result, the design of a competitive data processing system must proceed as a series of tradeoff decisions between various technological alternatives, considering their costs and the processing and functional power offered by each option. These tradeoffs are made in nearly all stages of development and at all levels of detail. As the variety of technological alternatives available for a given function increases, the cost-performance decisions gain in importance and often override purely technical considerations. A classic example is the advent of storage hierarchies, which combine the favorable features of different technologies to provide high-speed, low-cost storage systems.

Presented in this paper is a technique for the rapid evaluation of technological alternatives in storage hierarchies. We assume that the processor and storage hierarchy of a computer system can be represented by a closed queuing network where each stage in the network represents a different level in the storage hierarchy. Thus our purpose is to present an efficient method of evaluating storage hierarchies and not to discuss a new model.

Figure 1 A storage hierarchy



Exact solutions to the equilibrium joint probability distribution of queue lengths exist for some closed queuing networks.[1,2] Their computational requirements, however, are excessive for hierarchy evaluation.[3] Therefore, we use an approximate solution to the steady-state condition of the closed queuing system. This approximation has a computational growth rate that is linear—computationally efficient—in the number of network stages.

We have successfully correlated the results of this approximation technique with results obtained from both simulation of the cyclic queuing system and theoretically exact models.

A storage hierarchy is a convenient way of representing the multitude of storage devices used in computer systems. Each level of the hierarchy contains devices of different physical characteristics. A typical hierarchy is shown in Figure 1. Higher-level devices have higher cost, lower access time, and lower storage capacity. A properly designed hierarchy, as a whole, ideally displays the favorable features of its elements and behaves as a low-cost, fast-access time, large-capacity storage device. This effect derives from a general property of programs called "locality of reference," and can be expressed as follows: If two adjacent storage device levels have capacities C_i and C_{i+1}, and each has frequencies of data requests f_i and f_{i+1}, then the concept is expressed as follows:

$$\frac{C_i}{C_{i+1}} < \frac{f_i}{f_{i+1}}$$

One can say that the higher-speed storage tends to act like a cache for the lower levels of the hierarchy.

Levels of a hierarchy are divided into two groups by a *delay boundary*. The access time of devices at levels below the delay boundary are large enough to warrant the suspension of processes that generate requests to these devices and to switch the CPU to another process. Requests above the delay boundary (at higher levels) do not typically cause process switching.

Storage hierarchies constitute the hardware base for virtual storage in System/370. The System/370 virtual storage contains up to three hierarchical levels (cache, main storage, and disk). It is possible to extend the hierarchy concepts to automatically manage more levels than have been in the past. Effects of virtual storage have been included in this study by the choice of miss ratio curve,[4] discussed later in this paper.

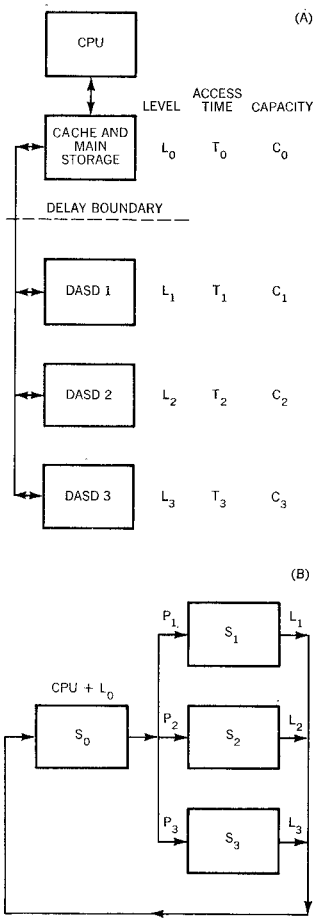
In this paper, we wish to show the development of a set of evaluation tools suitable for solving problems in the following areas of hierarchy design:

- Selection of devices from a set of alternative storage technologies for each of the levels of the hierarchy. The available technologies are called the "technology menu."
- Determination of the optimum number of hierarchical levels and level capacities.
- Evaluation of the effects of variations of hierarchy parameters on system performance. Significant conclusions should be insensitive to expected variations of input parameters.
- Assessment of the cost dependence of a technology on its total production volume.

We begin by representing a computing system by means of a cyclic queuing model.[4] The list of system parameters required by the model is given. The succeeding section deals in more detail with the queuing model and derives a fast approximative solution for the utilization of its components that is based on steady-state flow equations. A programmed version of this model is described later as a vehicle for solving the previously discussed design problems. Two different criteria of system optimality are used, one for interactive systems and the other for batch-oriented systems. An Appendix is provided in which we correlate the approximative solution of the queuing model with known theoretical results, and point to some extensions of the method.

An extensive effort was made to correlate the outputs of an analytic model with those obtained from a simulation model of a cyclic queuing network. The simulation model was created to investigate general network queues with a variety of service time distributions and queuing disciplines. (A proof has also been worked out by which the average queue length at a given hier-

Figure 2 Relationship between storage hierarchy and cyclic queuing system. A. Storage hierarchy; B. Cyclic queuing system



archical stage is shown to have the same form as that given by Gordon and Newell,[1] Jackson,[2] and Buzen.[3])

Our verification effort proceeded first by correlating the results of the simulation model with results obtained from theoretical models.[1,5] We then compared the results obtained from the analytic model described here with comparable simulation results for a variety of situations, i.e., two- and three-level storage hierarchies, multiserver stages, and varying levels of multiprogramming. In total, more than forty comparisons were made. (Table 2 of the Appendix shows representative results obtained from these comparisons.)

The approach described in this paper assumes the validity of the verification results and also depends on the validity of the cyclic queuing model as a satisfactory representation storage hierarchies in actual use. We have investigated current computer systems with this model and have found that optimal configurations for given workloads generated by the model correlate well with actual configurations associated with the given workloads. The model that we now describe has also been used for the evaluation of new storage mechanisms.

A model of the CPU and storage hierarchy

The problems we have just outlined are basically those of optimization that require the evaluation and comparison of many alternative hierarchical configurations. This task is accomplished in two steps. We first give the description of a model that incorporates the hierarchy into a multiprogrammed system. This system and its workload are defined by using only a few significant parameters. The output of the model yields the utilizations of the system components, i.e., the fractions of busy times of the processor and the levels of the storage hierarchy. The second step—the inclusion of cost information, the derivation of cost-performance values from CPU utilization, and the organization of optimization procedures—is accomplished in the succeeding section.

the computing system

The framework for the subsequent hierarchy evaluations is a simple system consisting of a CPU coupled with storage hierarchy. The system is supposed to operate in a multiprogramming mode, with a fixed degree of multiprogramming m. All m processes are statistically equivalent, and there is no priority scheme involved in assigning resources to the processes. Once a process is assigned to the CPU, the process uses the CPU until a request is presented to a storage level below the delay boundary. At that time, a process switch occurs, and the CPU is as-

signed to the next process waiting in a CPU queue. If a reference occurs to a level above the delay boundary, no process switch takes place. In our system, all storage levels above the delay boundary are clustered into a single level L_0 (usually composed of the cache and main storage). Figures 2A and 2B show the correspondence between this system and the queuing model.

We now describe the parameters of the system and its workload in terms of the inputs required by the queuing model.

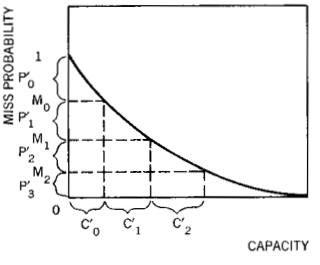
- Average access times in seconds $T_1 \cdots T_N$ and the structures of the hierarchical levels to be used for the arrival rate functions f_i, which are explained later in this paper. The access times include the seek and search action of the access mechanisms, the data transfer time to hierarchical level L_0, but not the waiting times if the access mechanism is busy.
- Capacities C_i in millions of bytes of the hierarchical levels.
- The level of multiprogramming m is equal to the number of processes in the queuing system.

miss ratio curves

We now introduce the concept of *miss ratio curves*.[6] Miss ratio curves represent a concise description of the workload of a system. They also include the effects of programming techniques and data usage. A miss-probability curve (plotted in Figure 3) is a graph of the probability M_0 that a storage reference from the central processor is not found in level L_0 versus the capacity of that level. A miss-ratio curve, given in Figure 5 in the Appendix, is calculated from the log of the complement of the hit ratio versus the log of the capacity.

The miss ratio curve is generated from the *reference string*, which is the sequence of pages that is referenced when a program is being executed. An equivalent curve is called the "hit ratio curve," i. e., the plot of the probability that a page reference is found in a buffer versus the buffer capacity, provided the page replacement algorithm is a stack algorithm.[3]

Figure 3 Example miss probability curve



Such a curve is a function of both the replacement algorithm used for storage management and the type of computing work load. As an approximation, we extend the interpretation of this curve to a representation of the distribution of references to all $N + 1$ levels of a hierarchy. The hit ratio p_i' is the probability that a storage reference is satisfied from level i. Knowing the effective capacities C_i', p_i' is illustrated in Figure 3 and can be obtained as follows:

$$p_0' = 1 - M_0$$

$$p_N' = M_{N-1}$$

and

$$p_i' = M_{i-1} - M_i \tag{1}$$

where

$i = 1, \cdots, N$

In this paper, we assume that a miss ratio curve corresponds to an individual process. All m processes have the same statistical behavior (i.e., the same curve). All storage levels except the last are divided into m equal parts, and a process has the effective capacity $C_i' = C_i/m$ at each level i.

In some systems, information used by a process may be shared with other processes as well. In such cases, the effective capacity is $C_i' \geq C_i/m$, and the extreme case is $C_i' = C_i$ when all programs and data in the system are shared concurrently by all m processes.

The average CPU instruction rate G is the average number of instructions per second executed under the assumption that all programs and data are found in level L_0. G is used to derive the mean CPU execution interval T_o, the time during which a process uses the CPU before it generates a request across the delay boundary. The mean execution time is given as follows:

$$T_0 = \frac{1}{2 \times G \times M_0}$$

where M_0 is the probability of a miss from level L_0, and the constant 2 reflects the assumption that there are two storage references per instruction.

The cyclic queuing model

The purpose of the cyclic queuing model is to determine the utilization of the CPU and storage levels of the computing system previously introduced. The model is composed of $N + 1$ stations $S_o, \cdots, S_N$, where S_0 is interpreted as the CPU of a single processor system and all storage above the delay boundary. Stations $S_1, \cdots, S_N$ represent the levels of a storage hierarchy below the boundary. A constant number m processes circulate in the system. When a task completes its CPU service interval, it leaves level S_0 and it is routed to $S_1, \cdots, S_N$ with probabilities $p_1, \cdots, p_N$. These are obtained from the hit ratios:

$$p_i = \frac{p_i'}{1 - p_0'}$$

where

$i = 1, \cdots, N$

and

$$\sum_{i=1}^{N} p_i = 1$$

After being serviced in a station, processes are returned to S_0. The transmission times are considered to be negligible; hence, every process must be in one of the stations $S_0, \cdots, S_N$ at any time. A process in any station is waiting for service or being serviced.

We denote Q_i as the average number of processes found in S_i (waiting or being served), and R_i is the average throughput rate i.e., the flow in processes per second). The following steady-state equations hold for the system:

$$R_i = p_i R_0 \tag{2}$$

where

$$i = 1, \cdots, N$$

and

$$\sum_{i=0}^{N} Q_i = m \tag{3}$$

Equation 2 is the flow conservation law, and Equation 3 shows that each process always exists in one of the stations. We can observe that Equation 3 determines the values of rates R_i up to a single multiplicative constant (i.e., it gives the ratios R_i/R_j, but not the absolute values). The missing condition can obtained from Equation 2.

service station

Each service station S_i can be viewed as a black box that is characterized by the dependence of Q_i on the rate R_i. We assume that this dependence is a function of $Q_i = f_i(R_i)$ and that functions f_i are known for all S_i stations in the system. In this view, a station S_i is analogous to a nonlinear resistance; S_i is analogous to a current; and processes Q_i are analogous to the voltage drop across the resistance.

The concept of describing S_i by means of f_i is admittedly an over-simplification, since Q_i depends not only on the structure of the station, but also on the pattern of arrivals of processes. The simplifying assumptions are as follows:

- Arrivals of processes at the stations are random.
- Arrival times are independent of previous departures from the same station.

Thus f_i describes station S_i in an open-loop Poisson-arrival situation. This is the essence of the approximation inherent in the

present analysis. The advantage of the black, box analogy is that it provides a uniform way of characterizing each station, regardless of the technological and structural features of its inner construction. All structural features of S_i relevant to this analysis should be projected into the nature of f_i. Now we can rewrite Equation 2 as follows:

$$Q_i = f_i(p_i \cdot R_0) \tag{4}$$

where

$i = 0, \cdots, N$

We can also rewrite Equation 3 as follows:

$$\sum_0^N Q_i = \sum_0^N f_i(p_i \cdot R_0) = f(R_0) \tag{5}$$

If f_i are continuous, monotonically increasing functions in R_i, then f is also continuous and monotonically increasing in R_0.

Equation 5 thus uniquely determines the value of R_0, and, through Equation 2, determines the values of all other throughputs R_i. Therefore, from a knowledge of m, functions f_i and probabilities p_i, it is possible to find the rates R_i and average number of processes Q_i in the service stations.

utilization factors and examples of stations

In reality, each service station has some of the internal organization of waiting queues, service disciplines, and servers. By *servers* we mean the active elements in a station e.g., the CPU or various access mechanisms and ports into storage. There may be one or more servers (of the same type) in any station. The time spent by a process in a station (i.e., the time between its arrival and departure) is composed of two parts: *waiting time* (time spent in queues, which may be zero), and *service time* (time the process keeps one of the servers busy). The average service time in S_i is denoted T_i. We define the *utilization factor* U_i of a server as the fraction of time the server is busy:

$$U_i = R_i T_i / D_i \tag{6}$$

where

$i = 0, \cdots, N$

and where D_i is the number of servers in S_i. All servers in a station are assumed to have the same utilization.

examples

Five examples are now given to illustrate the hierarchy modeling principles as they apply to the processes inside a service station.

Example 1. The service station is a single server with exponentially distributed service time and with mean value T. From Reference 7 the function f is then

$$Q = f = \frac{RT}{(1 - RT)}$$

This structure typically corresponds to a CPU with exponentially distributed execution intervals.

Example 2. There are D servers in a station with exponentially distributed service times (of mean value T). A separate queue is associated with each server, and incoming processes go to any queue with equal probability $1/D$. This service station may correspond to a disk facility with K actuators.

In this case, the function f is as follows:

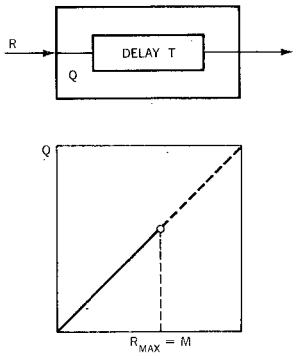
$$Q = f = \frac{DRT}{(D - RT)}$$

Example 3. The station shown in Figure 4 is acting as a delay element with mean delay T. Each process is serviced immediately upon its arrival, regardless of the other process being serviced at the same time. Note that, since there is no waiting queue, the utilization of the delay element may be greater than one. The behavior of the station is expressed by the following equation:

$$Q = RT$$

A disk subsystem with a separate actuator assigned to each process is representative of this structure.

Figure 4 Service station acting as a delay line



Example 4. In general, any analytic results available from queuing theory are applicable for expressing f (multiserver systems, nonexponential service times other than FIFO queuing disciplines). A particularly useful relation is the Khintchine-Pollaczek equation, given in Reference 7, for single server, general service time distribution $(M/G/1)$ systems:

$$Q = U + \frac{U^2(1 + C^2)}{2(1 - U)} \tag{7}$$

where C is the coefficient of variation of the service time and $U = RT$. This relationship is used in the programmed version of our method to allow for fitting the first two moments of empirically obtained service time distributions (such as for disk access times and CPU execution intervals).

Example 5. In cases where the complexity of a station (storage hierarchy level) does not permit the use of results from queuing theory, the function $Q_i = f_i(R_i)$ can be obtained by simulation. Such may be the case when the station contains a channel, several control units with sector queuing, and queue sorting. In general, functions f_i serve as a uniform means for defining the behavior of hierarchy levels.

The algorithm for solving Equation 5 for the queuing model equilibrium, depends on the way functions f_i are defined. Since a purely analytic solution is seldom possible, a simple binary search with less than ten iterations was adopted in the program used for optimizations. The result is a general solution with acceptable speed.

Optimal hierarchies

cost-performance

Since storage hierarchies are usually not end products, but, rather, components of data processing systems, it would make little sense to optimize an isolated hierarchy, regardless of the system context. We consider that a hierarchy is optimal if the system that incorporates it cannot be made better by changing the hierarchy parameters alone.

The goodness of a system as judged by a user is generally a complex mixture of his estimate of the raw processing power, system cost (purchase or rental), compliance with specified functional requirements (such as response time), and many intuitive factors (such as ease of operation, maintenance, etc.) Any attempt to quantize the goodness function is clearly an approximation; however, such formalization is useful and necessary to assist intuition, which can easily fail in complex situations. We shall call the *cost-performance function* CP and designate it to be any such quantitative expression. The system that yields the minimum value of CP is said to correspond to the best system. We shall use two representative CP functions: (1) for batch systems (CP_B); and, (2) for interactive systems (CP_I). The best batch system is one that performs a job most economically. We choose the cost-performance function for a batch-oriented system as follows:

$$CP_B = \frac{\text{system rental per second}}{\text{average number of instructions per second}}$$

$$= \frac{K' + \sum_{0}^{N} C_i K_i}{G U_0} \quad \text{dollars per instruction} \qquad (8)$$

Here K' is the CPU rental; C_i is the capacity of hierarchical level i in millions of bytes; K_i is the rental of level i; U_0 is the CPU utilization; and G is the execution rate of the CPU.

Interactive systems are required to meet given specifications of throughput rate S (in transactions per second) and response time E (in seconds). The better the interactive system, the lower the cost at which it meets these requirements. The cost-performance characteristic of an interactive system is expressed as follows:

$$CP_I = \frac{K' + \sum_{0}^{N} C_i K_i}{GU_0} \tag{9}$$

subject to the constraints

$$IS \geq GU_0 \tag{10}$$

where I is the required number of instructions per transaction

and

$$ES \geq m \tag{11}$$

Equations 10 and 11 require some further explanation. Equation 10 states that the CPU can maintain the desired processing power. The left side of Equation 11 is the average number of transactions inside the computing system waiting for response. This number must not be lower than the degree of multiprogramming m used by the model to obtain U_0. Also note that the number of transactions per second S is not equal to the rate R_0 of the queuing model. The processing of a transaction may require several CPU execution intervals.

optimization procedures

From the description of the queuing model, we know how the CPU utilization U_0 is linked with various system parameters. Thus for fixed parameters, one can calculate the cost-performance characteristic of the system by Equations 8 – 11.

For the purpose of solving various optimization problems such as those outlined in the introduction, an optimization program has been written that is based on an intelligent search. The program is composed of a system of nested loops, each one serving to change one parameter or class of parameters (such as C_i). The innermost loop contains a cost-performance calculation. The program returns the hierarchy configuration that yields the lowest value of CP. The various optimization problems require individualized parameter settings and search strategies. In general, only a subset of all parameters is varied for each run of the program, while the others remain constant.

Since the miss-ratio curve and CPU characteristics are included in the input values, it is possible to find optima over a range of systems and workloads. Thus the program goes beyond calculating optima for hierarchies in a fixed environment. In such cases, the program can be made to produce reports on the numbers of times a given technology has been included in the optimum hierarchy and the total volume (capacity) represented by these inclusions.

Particularly important areas of investigation are sensitivity studies aimed at determining the degree to which a particular

conclusion is invariant with variations of system parameters. One might ask, for example, whether a technology that has been found to be optimal for the second level of a hierarchy, for a range of applications is still optimal if its costs is increased by ten percent.

Concluding remarks

We have described a simple model of a computing system with a storage hierarcy, based on the view that such a system can be represented by a cyclic queuing model. Each stage in the system represents a component in the storage hierarchy with the exception of one stage that represents a combination of the central processor and main storage.

The average utilization and queue length of the stages are calculated by treating each stage as an open loop queue with random arrivals. Average queue length as a function of average throughput rate is represented either analytically or by some tabular relationship based on measurements.

We wish to emphasize again that the purpose of the model is not to provide a detailed evaluation of storage hierarchy performance. Rather, the goal in formulating this model has been to create a computationally efficient technique for evaluating the gross effects on performance of variations in computing workload and hierarchical components. The approximative nature of the model is consistent both with this goal and with the accuracy of the input data.

Appendix: Model validation and extensions

We now compare results obtained by using the analytic model with those obtained by using a simulation model of a cyclic queuing system and with available theoretical results. Also described are the following extensions to the model:

- Sharing data among tasks on levels of the hierarchy other than the archive level.
- Accounting for CPU overhead in switching processes.
- The use of a family of miss ratio curves to test the sensitivity of the optimal storage hierarchy (derived by the model) to variations in computing environment.

model validation

Since a number of approximations have been made in the formulation of the theoretical model, it is well to begin the model validation by comparing model results with theoretical results for a

Table 1 Comparison of theoretical and analytical modeling results

T_1	T_2	c_1	c_2	U_1 (*model*)	U_1 (*theory*)
0.25	1	1	0	0.249	0.250
0.25	1	1	1	0.246	0.243
0.25	1	2	0	0.249	0.247
0.50	1	1	0	0.490	0.499
0.50	1	1	1	0.481	0.482
0.50	1	2	0	0.488	0.465
0.90	1	1	0	0.807	0.821
0.90	1	1	1	0.784	0.756
0.90	1	2	0	0.788	0.713

T_1 service time of CPU cache main storage
T_2 service time of auxiliary storage
c_1, c_2 coefficients of variation of level i
U_1 utilization of level i

simple system so as to gain a measure of confidence in the model. The simple system is a two-level storage hierarchy with a multiprogramming level of four. Theoretical modeling results are summarized in Table 1. In five of the nine cases of input variation, the theoretical and modeling results are within one percent of each other. In three of the remaining four cases, the outputs differ by less than three percent. In one case, the difference is less than eight percent.

A simulation model of a cyclic queuing system has been developed to further investigate the validity or assumptions made in the analytic model. Both the analytic model and simulation model were modified to include stages with identical multiple queues. This modification allows us to analyze more accurately storage hierarchies with a multiplicity of independent data paths to a given level, as in the case of a direct access storage device with several arms.

Table 2 summarizes the results obtained from both models. A comparison of the average queue length and utilization for comparable stages in the cyclic queuing network demonstrates that the analytic model results correlate well with those of the simulation model. All stages in the model have exponentially distributed service times. Since our use of the model has been confined to exponentially distributed service times, correlations of the analytic model with other distributions have not been considered.

The purpose of the analytic model is to approximate efficiently the behavior of a computer system with a storage hierarchy. This validation effort demonstrates that answers generated by the model are consistent with this goal.

Table 2 Comparison of simulation and analytical modeling results

M	T_1	T_2	T_3	P_3	m_2	m_3	U_1	U_2	U_3	Q_1	Q_2	Q_3	*key*
18	160	29	207	0.254	8	2	0.995	0.017	0.164	17.8	0.130	0.370	A
							1.000	0.017	0.160	17.5	0.136	0.380	S
1	160	29	207	0.254	8	2	0.681	0.092	0.225	0.681	0.094	0.225	A
							0.681	0.088	0.226	0.681	0.088	0.226	S
4	160	29	207	0.254	8	2	0.975	0.017	0.161	3.59	0.107	0.295	A
							0.994	0.017	0.166	3.47	0.136	0.392	S
6	55	1000	---	---	15	---	0.242	0.321	---	0.300	5.7	---	A
							0.237	0.303		0.294	6.0		S
3	98	28	---	---	27	---	0.918	0.010	---	1.818	0.185	---	A
							0.967	0.010		1.722	0.270		S
2	79	28	1000	0.050	4	15	0.870	0.078	0.039	2.300	0.249	0.451	A
							0.929	0.077	0.040	2.044	0.328	0.610	S

M level of multiprogramming
T_1, T_2, T_3 average service time of level i (milliseconds)
P_3 probability of an access to level 3
m_2, m_3 number of servers of level i

U_1, U_2, U_3 utilization of servers of level i
Q_1, Q_2, Q_3 average queue length of level i
A analytic model result
S simulation model result

shared storage

The model, as described in the body of this paper, associates a fraction of the capacity of each level of the hierarchy (except the archival level) with a given process as follows:

where

$$\frac{C_i'}{C_i} = \frac{1}{m}$$

is the fraction of the capacity of level i associated with a task and m is the level of multipramming.

A more accurate view takes into account the sharing of data among processes. For example, the portion of main storage that is allocated to operating system programs is shared by all processes.

As mentioned in the first section of this paper, sharing can extend to the auxiliary storage, including the extreme case in which a process shares all auxiliary storage with other processes. The concept of the capacity of level i associated with a process is given by the following relationship:

$$C_i' = S_i C_i + \frac{(C_i - S_i C_i)}{m}$$

We note that the level of multiprogramming cannot become zero. Here S_i is the fraction of level i storage that is shared. The last (slowest) level of the hierarchy always has a value of $S_i = 1$ since the data base is held at this level. The effect of storage sharing is to increase the probability of finding requested information in shared storage.

As an illustration of this concept, consider a situation wherein four million bytes of main storage are available, and the level of multiprogramming m is 4. The resident operating system programs take up one million bytes of main storage. The probability of a miss from main storage is given as 10^{-4} if the operating system is not considered as shared among all processes, since each process has one million bytes associated with it. If the sharing concept is employed, the miss ratio is 2×10^{-5} because the process has one million of the operating system storage plus 0.75 million bytes [4MB − ¼(4MB − 1MB)] of private storage associated with it. This example illustrates that knowledge of the amount of shared storage at each level of the storage hierarchy is important for an accurate evaluation.

process switching

The concept of multiprogramming is used in computing systems to increase resource utiliztion by allowing many processes to use main storage concurrently. As a consequence, the execution of a process that requires data in auxiliary storage can be suspended and another process that is waiting to use the central processor can begin execution.

Multiprogramming does, however, impose added overhead on the central processor, since the process presently in execution must be suspended while a process waiting for the CPU is selected. If the average execution time associated with process switching T_s is known, the effect of this overhead can be accounted for in the following manner:

1. Increase the average time T_0 that a process is in execution by the process switching time.
2. Modify the CPU utilization generated by the model by the following rule:

$$\text{Effective CPU utilization} = \text{model CPU utilization} \times \frac{T_0}{T_S + T_0}$$

The effective CPU utilization then represents the percentage of time that the CPU is executing process instructions.

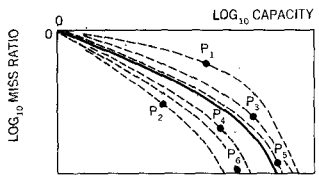
miss ratio curves

We now propose one solution to the problem of defining a computer workload by means of a miss ratio curve. If one knows from measurements the bounds on the distribution of storage references to the levels of the hierarchy, a useful technique for evaluating a hierarchy is to create a family of miss ratio curves

that represent these bounds. For example, assume that a miss ratio curve is given for a specific workload as shown by the solid line in Figure 5, and measurements show that the bounds in the miss ratio curve are represented by the set of points P_1 through P_6 in that figure. We then take as the family of curves a set of the general shape of the solid curve that pass through these points. The dashed curves of Figure 5 illustrate such a family of curves.

A set of modeling runs can then be made, one for each miss ratio curve, and the resulting hierarchies may be compared. Should vastly different hierarchy configurations result, then one has an indication that the combination of technologies that has been selected for the hierarchy is very sensitive to variations in workloads, and a carefully designed measurement of workload is required. Conversely, minor variations in hierarchy configurations with variations in miss ratio curves lead to the conclusion that the hierarchy is relatively insensitive to the extremes of the expected workload.

Figure 5 A family of miss ratio curves



CITED REFERENCES

1. W. J. Gordon and G. S. Newell, "Closed queuing systems with exponential servers," *Operations Research* **15,** 2, 254–265 (April 1967).
2. J. R. Jackson, "Jobshop-like queuing systems," *Management Science* **10,** 1, 131–142 (October 1963).
3. J. P. Buzen, "Computational algorithm for closed queuing networks with exponential servers," *Communications of the ACM* **16,** 9, 527–531 (September 1973).
4. I. L. Traiger and R. L. Mattson, *The Evaluation and Selection of Technologies for Computer Storage Systems,* IBM Research Report RJ 967, February 1972, may be obtained from the IBM Research Laboratory, Monterey and Cottle Roads, San Jose, California 95193.
5. G. S. Shedler, "A queuing model of the multiprogrammed computer system with a two-level storage system," *Communications of the ACM* **16,** 3–10 (January 1973).
6. R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal* **9,** 2, 78–117 (1970).
7. L. Takacs, "A single-server queue with Poisson input," *Operations Research* **10,** 3 (1962).