Y. Bard

An Analytic Model of the VM/370 System

Abstract: This paper describes an analytic model of an interactive multiprogrammed computer system. The model accepts a multiple-user-class, transaction-oriented workload description and a system configuration description, and it produces predictions of resource utilizations, transaction rates, and average transaction response times. The solution method involves nearly complete decomposition, with a closed queuing network representing the multiprogrammed set. Asymptotic formulas are used to generate good initial guesses for an overall iterative scheme. Extensive validation results are presented.

1. Introduction

This paper describes a model [1] of a virtual memory, multiprogrammed, interactive computer system called VM/370 [2]. A previous model of the system assumed a homogeneous (single class) user population and concentrated most of its computational efforts on determining the distribution of multiprogramming levels [3]. Neilson [4] extended that model to accommodate two user classes—batch and interactive. The present model, while an outgrowth of the previous ones, differs from them in almost all details. It accommodates multiple user classes whose workload descriptions for an existing system are derivable directly from available measurement facilities [5]. Like the preceding ones, our model employs the principle of nearly complete decomposition [6] to separate the transaction flow outside the CPU from the actual processing phase, the latter being treated by means of a closed queuing network model. Unlike the preceding ones, the present model iterates between the overall flow equations and the queuing network equations. The distribution of multiprogramming levels is no longer evaluated, only average values being required. Simplified asymptotic formulas significantly reduce the processing times for large problems involving many user classes and/or a high multiprogramming level.

The model has been thoroughly validated against live workloads in a large variety of installations and has been used extensively for predicting the performance of various VM/370 configurations under many different workloads.

The paper is organized as follows: To provide the rationale for the structure of the model, we describe first the VM/370 scheduler (Section 2). Model inputs are provided in the form of workload (Section 3) and configuration (Section 4) descriptions. The workload description may need to be transformed to the specified configuration (Section 5). The flow of transactions through the system is modeled by means of Little's formula (Section 6), but the details of the flow model depend on whether FIFO (Section 7) or CPU fair share (Section 8) scheduling is employed. The overall system model (Section 9) consists of iterative alternate applications of the flow model and the multiprogrammed-set model (Section 10). The latter may be approximated by means of asymptotic formulas (Section 11), which are valid at high multiprogramming levels.

The special case of a system with an auxiliary processing unit is treated in Section 12. Once convergence has been attained, the model computes predictions for the usual performance factors (Section 13), such as response times and resource utilizations. The model has been validated extensively (Section 14), both against live workloads and against controlled benchmarks.

2. VM/370 scheduler

Figure 1 illustrates the manner in which VM/370 schedules its work. A user is *dormant* when he has no work outstanding. Time spent in the dormant state is called *think time*. When a user enters a request for service, he is first placed in the *eligible set*, and then, when system resource availability permits, he is placed in the *multiprogrammed set*. Only members of the latter set actually receive CPU time and I/O services and are allocated space in main storage. The multiprogrammed set consists of two subsets: Q1, containing users who have received less than

a certain amount of CPU time (about 100 ms) since leaving the dormant state, and Q2, consisting of all others. Similarly, the eligible set is broken up into Q1 and Q2 candidates. The total number of users in the multiprogrammed set is the multiprogramming level (MPL).

The system estimates the size of each user's *working set*, which is the amount of main storage space the user's programs are thought to require in order to run efficiently. A user is admitted into the multiprogrammed set only if sufficient main storage space is free to accommodate his working set, with Q1 candidates taking precedence over Q2 candidates.

The VM/370 scheduler admits Q2 candidates in FIFO order. The order may be modified by means of external user priorities and scheduler bias factors [7]. The latter are designed to penalize heavy users of bottlenecked system resources.

An alternative scheduler is available [8], which schedules admissions so as to equalize, as far as possible, the share of CPU time obtained by different users. Specific users may be given more or less than their *fair share* according to external priorities, and scheduler bias parameters may be used to include resources other than the CPU in the fair share calculations.

The model described below can accommodate the following scheduling disciplines:

1. Strict FIFO.
2. CPU fair share, with or without priorities.

With very slight modifications, multiple-resource fair share could also be treated.

The VM/370 scheduler also permits dedication of specific system resources (e.g., a given fraction of CPU time or a certain number of main storage page frames) to given users. The model handles such users by subtracting the dedicated resources from the pool of resources available to other users. In the case of a dedicated percentage of CPU time, the model treats the CPU as though it were slowed down by an appropriate factor. For instance, a 50 percent dedicated CPU appears to the model as a CPU whose speed has been cut in half.

3. Workload characterization

It is convenient to break up each user's workload into *transactions*. A transaction is defined as the work done for the user from the time he becomes a Q1 candidate until the next time that he becomes either dormant or a Q1 candidate again, whichever occurs first. Two types of transactions are recognized:

Trivial transaction: One that is completed in Q1.

Nontrivial transaction: One that requires at least one stay in Q2.

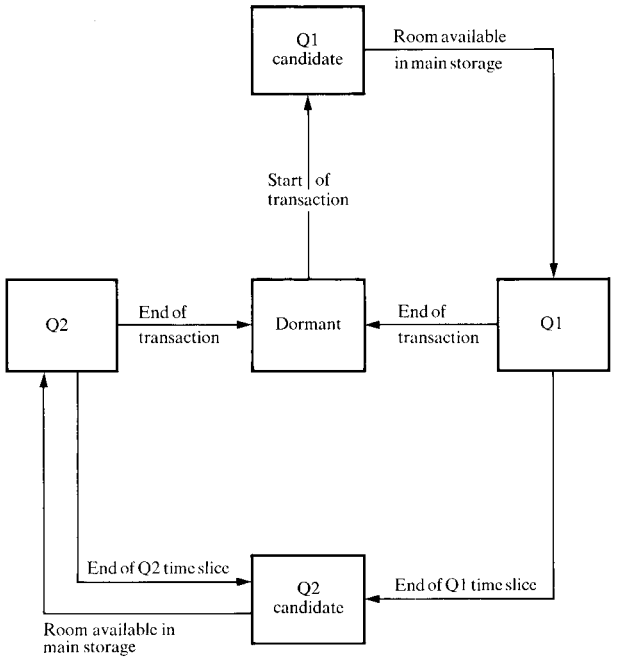

Figure 1 Transaction flow in VM/370.

Some users (typically batch virtual machines) may present to the system a *continuous workload*, wherein they cycle indefinitely between the Q2 and Q2 candidate states. A continuous workload may be regarded as a sequence of nontrivial transactions, each requiring one Q2 slice of CPU time, separated by zero think times.

The model accepts the following workload characterization: The entire user population may be broken up into several user classes. Although it is desirable that users within a class have a more or less homogeneous workload, good results have been obtained in many cases with all users lumped into a single class. Each user class belongs to one of the following types:

T: Trivial transactions only.

N: Nontrivial transactions only.

B: Both trivial and nontrivial transactions.

C: Continuous workload.

From remarks made above, it is evident that a C class is a special case of an N class and requires no further considerations.

The average values of the following quantities are required for each user class:

- Number of logged-on users.
- Secondary storage paging slots occupied per logged user.
- Ratio of trivial to nontrivial transactions (B type only).

- Secondary storage data areas owned by class members, and access characteristics, including seek distances, block lengths, and relative access rates.

The average values per transaction of the following quantities are required for each transaction type within each user class (henceforth referred to as *transaction class*):

- Virtual CPU time (measured on a specified CPU model).
- Overhead CPU time (measured on the same CPU model).
- Working set.
- Page reads (measured in the same environment as the overhead CPU time).
- Paging index (an empirical, environment-independent measure of paging activity; see [5] for precise definition).
- Virtual input-output operations of various kinds (disk, tape, unit record).
- Think time.

A fuller description of the workload characterization and how it is obtained from measurements on existing systems can be found in [5].

4. Configuration description

Since the VM/370 System can run on almost any IBM System/370 configuration, the model must be provided with the following data describing that configuration:

- CPU model.
- Main storage size.
- Number and type of direct access storage devices and their placement on I/O channels.
- Placement of primary and secondary paging areas on the devices.
- Placement of each user class's data areas on the devices.

5. Workload conversion

Since the workload may have been measured on a system different from the one to be modeled, certain conversions may be required:

- CPU times are adjusted by the ratio of CPU speeds. Different ratios may be applicable to virtual and overhead times. These factors have been determined empirically.
- Number of page reads is adjusted according to the empirical model described in [5].
- CPU overhead time is adjusted to reflect the changed number of page reads.
- The distribution of users' page slots among the specified paging areas is determined according to the VM/370 allocation strategy [9]. Page migration from fast to slow paging devices can be modeled by assigning a disproportionately high fraction of page reads and writes to the fast devices.

6. Transaction flow model

For modeling purposes, it would be convenient if each stay in Q2 could be considered as a full nontrivial transaction. Fortunately, this is easily achieved, because to the model a transaction requiring T seconds of think time, t seconds of CPU time, and M stays in Q2 is equivalent to M transactions, each requiring T/M seconds of think time, t/M seconds of CPU time, and a single stay in Q2. The following assumptions are also made:

1. The Q1 stay at the start of a nontrivial transaction may be lumped into the succeeding Q2 stay.
2. Wait time for admission to Q1 is negligible; the scheduler diagram (Fig. 1) is thereby transformed into the simpler transaction flow model represented by Fig. 2.

The formulation of the model would be further simplified if there were a separate user class associated with each transaction class. This requires splitting each B-type user class with N members into two classes: A T-type with N_1 members and an N-type with N_2 members, such that $N = N_1 + N_2$ and $N_1/N_2 = RT_1/T_2$, where R is the ratio of trivial to nontrivial transactions and T_1 and T_2 are the cycle (think + response) times of the trivial and nontrivial transactions, respectively. While T_1 and T_2 are not known *a priori*, they are available at each iteration (see below), so that current values of N_1 and N_2 can always be calculated. In the sequel it is assumed then that the total user workload is divided into several classes, of which some enter only trivial transactions, whereas the others enter only nontrivial transactions.

Let $T_{i,1}$, $T_{i,2}$, and $T_{i,3}$ denote, respectively, the average time spent by a class i transaction in the dormant, eligible, and multiprogrammed sets, respectively, and let $N_{i,1}$, $N_{i,2}$, and $N_{i,3}$ be the average number of class i members in those sets at any one time. Let I_1 and I_2 denote the sets of indices corresponding to trivial and nontrivial transaction classes, respectively. The following equation is obvious:

$$N_{i,1} + N_{i,2} + N_{i,3} = N_i \qquad (i \in I_1, I_2), \tag{1}$$

where N_i is the total average number of class i members (N_i need not be an integer). The Q-admission policy requires that

$$S_1 + S_2 \leq S, \tag{2}$$

where S is the total available main storage capacity, and S_1 and S_2 are the average amounts of main storage occupied by trivial and nontrivial transactions, respectively. Clearly,

$$S_j = \sum_{i \in I_j} N_{i,3} W_i \qquad (j = 1, 2), \tag{3}$$

where W_i is the class i transaction average working set size.

According to Little's formula [10], the average number of transactions in each set is proportional to the average time spent by a transaction in that set. Thus,

$$N_{i,k} = \lambda_i T_{i,k} \qquad (i \in I_1, I_2;\ k = 1, 2, 3), \tag{4}$$

where λ_i is the class i transaction flow rate. Summing the above equations over k, we find

$$N_i = \lambda_i (T_{i,1} + T_{i,2} + T_{i,3}) \qquad (i \in I_1, I_2). \tag{5}$$

Furthermore,

$$\frac{N_{i,k}}{N_i} = \frac{T_{i,k}}{T_{i,1} + T_{i,2} + T_{i,3}} \qquad (i \in I_1, I_2;\ k = 1, 2, 3). \tag{6}$$

By assumption, $T_{i,2} = 0$ for $i \in I_1$. Hence, from (6),

$$N_{i,3} = \frac{T_{i,3} N_i}{T_{i,1} + T_{i,3}} \qquad (i \in I_1), \tag{7}$$

and

$$S_1 = \sum_{i \in I_1} \frac{T_{i,3} N_i W_i}{T_{i,1} + T_{i,3}}. \tag{8}$$

Before proceeding, we must take stock of which variables are known and which must be calculated. Specifically, the following are given in the workload description:

N_i, $\quad i \in I_1, I_2$: users in class i;

W_i, $\quad i \in I_1, I_2$: class i working set;

$T_{i,1}$, $\quad i \in I_1, I_2$: class i think time.

Furthermore, S (main storage capacity) is given in the configuration description. We shall also assume for the time being that the $T_{i,3}$ are known for all $i \in I_1, I_2$. We can now evaluate S_1, using (8). If it turns out that $S_1 \geq S$, then the system is entirely saturated with trivial transactions, and no further analysis takes place. If $S_1 < S$, then the storage requirement for nontrivial transactions is computed on the assumption that $T_{2,i} = 0$, $i \in I_2$, i.e., that nontrivial transactions are admitted into Q2 immediately. Analogously to (8), this storage requirement is

$$S_2^* = \sum_{i \in I_2} \frac{T_{i,3} N_i W_i}{T_{i,1} + T_{i,3}}. \tag{9}$$

Now, if $S_2^* \leq S - S_1$, it follows that the assumption $T_{i,2} = 0$ for $i \in I_2$ is acceptable. If $S_2^* > S - S_1$, storage is saturated, so that $S_2 = S - S_1$, or

$$\sum_{i \in I_2} \frac{T_{i,3} N_i W_i}{T_{i,1} + T_{i,2} + T_{i,3}} = S - S_1. \tag{10}$$

The solution of this equation depends on the system scheduling discipline, as shown in the next two sections. One could easily model constraints on resources other than main storage by introducing appropriate inequalities analogous to (2).

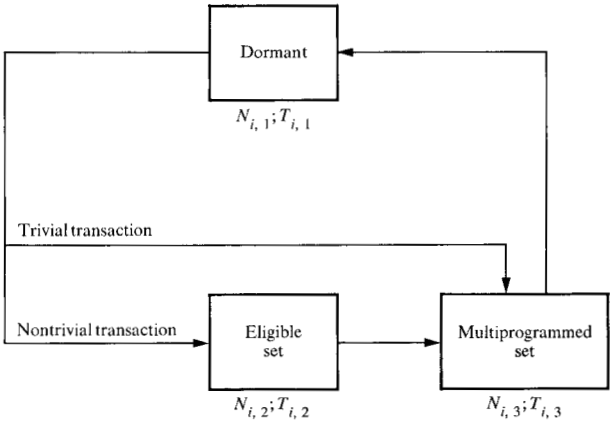

Figure 2 Simplified transaction flow.

7. FIFO scheduler

The FIFO scheduler treats all classes equally, so that expected Q2 admission wait time should be about the same for all. Hence, we may assume

$$T_{i,2} = Q \qquad (i \in I_2), \tag{11}$$

where Q is an unknown constant. Substituting in (10) we find

$$\sum_{i \in I_2} \frac{T_{i,3} N_i W_i}{T_{i,1} + Q + T_{i,3}} = S - S_1. \tag{12}$$

This is a single equation in the single unknown Q, which may be solved, say, by means of the Newton-Raphson method.

8. Fair share scheduler

The fair share scheduler guarantees equal amounts of CPU time to all CPU-bound users. Let t_i $(i \in I_2)$ be the CPU time required per transaction. This quantity is given in the workload description, but needs to be adjusted for B-class users to include the contribution from the trivial transactions generated by these users. The CPU consumption rate for each class i user is

$$U_i = \frac{\lambda_i t_i}{N_i}. \tag{13}$$

Substituting for λ_i from (5), we find that $U_i = t_i/(T_{i,1} + T_{i,2} + T_{i,3})$. Equating all U_i to a common "fair share" value U, we obtain the equation

$$T_{i,1} + T_{i,2} + T_{i,3} = \frac{t_i}{U} \qquad (i \in I_2), \tag{14}$$

which, when substituted in (10), yields

$$\sum_{i \in I_2} \frac{T_{i,3} N_i W_i U}{t_i} = S - S_1,$$

or

$$U = \frac{S - S_1}{\sum_{i \in I_2} \frac{T_{i,3} N_i W_i}{t_i}}, \tag{15}$$

and, from (14),

$$T_{i,2} = \frac{t_i}{U} - T_{i,1} - T_{i,3} \qquad (i \in I_2). \tag{16}$$

It may turn out that (16) gives negative values for some $i \in I_2$. This indicates that the corresponding user classes are not CPU bound. Let J be the set of indices for which (16) is negative. We then assign

$$T_{i,2} = 0 \qquad (i \in J)$$

and modify (15) to compute a new value of U:

$$U = \frac{S - S_1 - \sum_{i \in J} \frac{T_{i,3} N_i W_i}{T_{i,1} + T_{i,3}}}{\sum_{i \in I_2 - J} \frac{T_{i,3} N_i W_i}{t_i}}. \tag{17}$$

If resources other than the CPU are to be fair shared, only a redefinition of the t_i is required. In particular, the t_i may be defined as a weighted sum of different resources used per transaction.

The fair share scheduler permits user priorities to be specified. These priorities are interpreted by the scheduler as relative CPU utilizations [8]. That is, a user with a higher than average priority is given a larger than fair share of CPU time. These priorities are modeled as follows: Let b_i be the relative CPU utilization assigned to members of user class i. Hence, instead of having $U_i = U$ for all $i \in I_2$, we now have $U_i = b_i U$, and (15) must be modified to

$$U = \frac{S - S_1}{\sum_{i \in I_2} \frac{T_{i,3} N_i W_i b_i}{t_i}}. \tag{18}$$

Equation (16) becomes

$$T_{i,2} = \frac{t_i}{b_i U} - T_{i,1} - T_{i,3}, \tag{19}$$

and (17) is also modified correspondingly.

If main storage is large enough to admit all transactions immediately, the fair share scheduler controls CPU time allocation by assigning to users in the multiprogrammed set dispatching priorities inversely related to their CPU utilizations. Users generating trivial transactions at a very rapid rate may also become subject to the fair share restrictions. Since the model handles fair share scheduling only through control of eligible-set time, both of these effects are not modeled properly.

9. Overall system model

Once the $T_{i,2}$ are computed, all other quantities are easily determined. In particular, we obtain

$$N_{i,3} = \frac{T_{i,3} N_i}{T_{i,1} + T_{i,2} + T_{i,3}} \qquad (i \in I_1, I_2). \tag{20}$$

Recall that $N_{i,3}$ is the average number of class i members who are in the multiprogrammed set. When in that set, users receive CPU bursts, execute I/O operations, etc. These events occur frequently, on a milliseconds time scale, while Fig. 2 state transitions occur relatively infrequently, on a seconds time scale. Hence, to the rest of the system, the multiprogrammed set appears to be in an equilibrium state. The system may, therefore, be decomposed [6], and the multiprogrammed set may be treated as a closed queuing network with a constant user population $N_{i,3}$ $(i \in I_1, I_2)$. A suitable queuing network model (see Sections 10 and 11) may be used to find the steady state solution for the network. This solution yields estimates for the network residence times, which are the $T_{i,3}$. We have, therefore, the makings of an iterative procedure:

1. Assume initial guesses for the $T_{i,3}$.
2. Apply the transaction flow model to compute the $T_{i,2}$ and $N_{i,3}$.
3. Apply the queuing network model to recompute the $T_{i,3}$.
4. Return to step 2.

The cycle is broken when values of $T_{i,2} + T_{i,3}$ from successive iterations do not differ significantly. Although we have no convergence proof, experience with hundreds of practical cases shows that convergence generally takes place in two to ten iterations, where as initial guesses we take $T_{i,2} = 0$ and $T_{i,3}$ to be equal to the CPU time plus the sum of all channel data transfer times for all I/O operations generated by a class i transaction. To guard against the possibility of infinite looping, if convergence has not been attained after thirty iterations, the program terminates with a warning message.

10. Multiprogrammed set model

The multiprogrammed users are cycled among a set of queues: CPU queue, channel queues, and device queues. We assume that no user can overlap his own I/O and CPU operations. With suitable assumptions on service disciplines at the various queues, and ignoring the interactions between channel and device queues, one could solve this queuing network problem provided one only knew the total service time required by each transaction class at each queue [11].

Unfortunately, if one does ignore these interactions, invalid solutions may be obtained. An I/O operation to a direct access device consists of seek, rotational delay, and data transfer phases. During the first phase, only the device is busy (although the phase cannot be initiated without a brief access to the channel). During the second phase, the channel too is busy if there is no rotational position sensing. During the third phase, both channel and device are busy. These partial overlaps cannot be treated within the framework of classical queuing network theory, but the following four approximate methods can be suggested:

1. Ignore the overlap, and assign to each channel and device separately their full service times. Result: inflated response time prediction, since the overlapped service periods are counted twice.
2. Ignore the devices, and assign the entire service time to the channels. Result: inflated response time prediction, since overlap between seeks on different devices is not allowed.
3. Ignore the channels, and assign the entire service time to the devices. Result: underestimated response time, since contention and serialization of channel time are ignored.
4. Assign to the devices the phases when only they are busy, and to the channels assign the time when both channel and device are busy. Result: response time may be underestimated, since solutions may be obtained in which true device utilization exceeds 100 percent. Still, of the four possibilities, this one is by far the best [12].

This approach was indeed applied in the first version of this model. The convolution algorithm of [13] was used, requiring on the order of

$$(1 + M)L \prod_i (N^*_{i,3} + 1) \text{ operations,}$$

where M is the number of channels + number of devices, L is the number of transaction classes, and $N^*_{i,3} = \lceil N_{i,3} \rceil$ is the smallest integer containing $N_{i,3}$ (recall that since $N_{i,3}$ is an average value, it need not be an integer).

Subsequently, an approach was introduced that solved the problem of channel-device overlap and also considerably reduced the amount of computation required. There exist in the literature several open-queue models of the channel-device configuration [14–16] which explicitly model the complex interaction between these units. The models assume that I/O requests arrive in a Poisson stream, and they require as inputs the record lengths, placement of data sets on devices, and access rates to all data sets. The access rates are computed as follows: Let $z_{i,j}$ be the number of I/O operations requested by a class i transaction from data set j. According to (4), the class i transaction rate is $\lambda_i = N_{i,3}/T_{i,3}$. Hence, the access rate to data set j is

$$\sum_i z_{i,j} N_{i,3}/T_{i,3}.$$

The I/O submodel produces as outputs the average response times D_j to an I/O request from data set j. The total average delay suffered by a class i transaction due to its I/O requests is

$$\theta_i = \sum_j z_{i,j} D_j.$$

But the same average delay would be suffered by the transaction if all the channel and device queues were replaced by a single infinite-server queue with service time θ_i. Thus, the $(M + 1)$-queue original network may be replaced by a two-queue network, consisting of a single-server CPU queue and an infinite-server I/O queue. The number of operations in the convolution algorithm is thereby reduced to the order of

$$2L \prod_i (N^*_{i,3} + 1).$$

The number of operations in the solution of the open queuing I/O submodel is independent of the size and nature of the user population.

The convolution algorithm [13] is used to generate the queue length probability generating function for the two-queue network. An L-dimensional array $\mathbf{G}$ is initialized to

$$\mathbf{G}(j_1, j_2, \cdots, j_L) = \prod_{i=1}^{L} \frac{\theta_i^{j_i}}{j_i!} \quad (j_i = 0, 1, \cdots, N^*_{i,3}). \tag{21}$$

These quantities are proportional to the probabilities of queue lengths $j_1, j_2, \cdots, j_L$ at the "infinite server" I/O subsystem. The array $\mathbf{G}$ is transformed by applying the following feedback filter representing the CPU queue:

$$\mathbf{G}(j_1, j_2, \cdots, j_L) \leftarrow \mathbf{G}(j_1, j_2, \cdots, j_L) + \sum_{(i|j_i>0)} t_i \mathbf{G}(j_1, \cdots, j_i - 1, \cdots, j_L), \tag{22}$$

where, as before, t_i is the CPU time per class i transaction. Equation (22) must be applied in order of increasing values of the indices. The expected time in the multiprogrammed set for class i transactions with a population of $N^*_{i,3}$ is now given by

$$T^*_{i,3} = \frac{N^*_{i,3} \mathbf{G}(N^*_{1,3}, N^*_{2,3}, \cdots, N^*_{L,3})}{\mathbf{G}(N^*_{1,3}, \cdots, N^*_{i,3} - 1, \cdots, N^*_{L,3})}. \tag{23}$$

Similar equations hold for any other integral user population not exceeding $N^*_{i,3}$. To determine average responses $T_{i,3}$ at the (possibly) nonintegral average popu-

Table 1 Comparison of asymptotic and finite multiprogrammed set models, with $t_i = 2, 1$ and $\theta_i = 1, 1$.

No. of users in multiprogrammed set		*Time in multiprogrammed set Finite model [Eq. (23)] Asymptotic model [Eq. (29)]*	
$N_{1,3}$	$N_{2,3}$	$T_{1,3}$	$T_{2,3}$
1	1	4.00 3.56	2.67 2.28
2	2	7.44 7.53	4.37 4.27
3	3	11.43 11.52	6.31 6.26
4	4	15.45 15.52	8.30 8.26
5	5	19.46 19.51	10.29 10.26
10	10	39.48 39.51	20.27 20.25
4	1	9.78 9.82	5.50 5.41
8	2	19.79 19.81	10.45 10.40
16	4	39.79 39.80	20.42 20.40

lation $N_{i,3}$, a set of $L + 1$ integral populations surrounding $N_{i,3}$ is selected, the corresponding responses are computed, and $T_{i,3}$ is estimated by interpolation.

11. Asymptotic multiprogrammed set model

The computational and storage demands of the algorithm described in the preceding section grow rapidly as the number of transaction classes and the multiprogramming level increase. It is, therefore, of interest to determine what happens as the $N_{i,3}$ increase beyond bounds and whether such asymptotic results can provide useful approximations in practical cases.

Before proceeding, we define the following quantities:

$N = \Sigma_i N_{i,3}$ is the total multiprogramming level.

$a_i = N_{i,3}/N$ is the fraction of class i transactions in the multiprogrammed set.

x_j = fraction of multiprogrammed transactions in the jth server queue.

$\tau_{i,j}$ = average jth server service time for a class i transaction.

$f_j(n_j)$ = jth server service rate when there are n_j transactions in its queue. Clearly, on the average, $n_j = x_j N$.

Assume that N approaches infinity in such a way that the a_i remain constant (i.e., all transaction classes increase proportionately). Then, it has been shown [17] that, for a network satisfying the separability criteria with processor sharing servers, the x_j approach values which can be computed by using the following iterative scheme:

$$x_j^{(v+1)} = x_j^{(v)} \sum_i \frac{a_i \tau_{i,j}}{f_j(x_j^{(v)} N) \sum_k \frac{\tau_{i,k} x_k^{(v)}}{f_k(x_k^{(v)} N)}}, \tag{24}$$

where the superscript v refers to the iteration number. The iterations may be started with any set of positive x_j. In our case, we have two servers:

$j = 1$ is the single server CPU, with $\tau_{i,1} = t_i$ and $f_1(n_1) = 1$.

$j = 2$ is the infinite server I/O subsystem, with $\tau_{i,2} = \theta_i$ and $f_2(n_2) = n_2$.

Substituting these values into (24), we find after simplification:

$$x_1^{(v+1)} = x_1^{(v)} \sum_i \frac{a_i t_i}{t_i x_1^{(v)} + \frac{\theta_i}{N}},$$

$$x_2^{(v+1)} = \sum_i \frac{a_i \theta_i}{N\left(t_i x_1^{(v)} + \frac{\theta_i}{N}\right)}.$$

It is clear that meaningful results can be obtained only if it is assumed that the θ_i increase in proportion with N. Hence, let $\beta_i = \theta_i/N$ be assumed constant. Then

$$x_1^{(v+1)} = x_1^{(v)} \sum_i \frac{a_i t_i}{t_i x_1^{(v)} + \beta_i}, \tag{25}$$

$$x_2^{(v+1)} = \sum_i \frac{a_i \beta_i}{t_i x_1^{(v)} + \beta_i}. \tag{26}$$

It is now possible to apply the iterative scheme (25) by itself, then substitute x_1 in (26) to compute x_2 directly (or, more simply, set $x_2 = 1 - x_1$). We shall show, however, that even (25) need not be used in some cases. From (25) it is evident that

$$x_1^{(v)} < x_1^{(0)} \left[\sum_i \frac{a_i t_i}{\beta_i}\right]^v.$$

Therefore, if $\Sigma_i a_i t_i/\beta_i < 1$, then

$$x_1 = \lim_{v \to \infty} x_1^{(v)} = 0.$$

We now compute the average time $T_{i,3}$ spent by a class i transaction in the multiprogrammed set when N is large.

The time spent in the I/O subsystem is θ_i, and the CPU service time is t_i. Hence

$$T_{i,3} \geq \theta_i + t_i. \tag{27}$$

When $x_1 = 0$, CPU queuing time is negligible compared to $\theta_i = N\beta_i$, and (27) holds with equality sign. If $x_1 > 0$, then the CPU queue contains, on the average, Nx_1 transactions, and, for large N, is almost never empty. Hence CPU utilization is 100 percent, and the CPU utilization per transaction in queue is $1/Nx_1$. To obtain t_i seconds of CPU time, a transaction must spend t_iNx_1 seconds in the CPU queue. Therefore,

$$T_{i,3} = \theta_i + t_iNx_1, \tag{28}$$

but, because of (27), this result can be valid only for $Nx_1 \geq 1$. We may combine this remark with Eq. (28) into the single result

$$T_{i,3} = \theta_i + \max(t_i, Nx_1 t_i). \tag{29}$$

Equation (29) gives remarkably accurate results even for fairly small values of N, as illustrated in Tables 1 and 2, and its computation is much faster than (23). In view of this, the following strategy has been adopted for the model:

1. Apply the overall iterative scheme of Section 9 using Eq. (29) in step 3 until convergence is obtained. The convergence criterion used is that no value of $T_{i,2} + T_{i,3}$ has changed by more than 5 percent from one iteration to the next.
2. If storage capacity is insufficient to contain the entire G array, or if $N > 15$, terminate. (The value 15 was chosen somewhat arbitrarily, but experience has shown that Eq. (29) is generally accurate for $N > 15$, and Eq. (23) is not too costly to evaluate for $N \leq 15$.) Otherwise, go to step 3.
3. Continue the iterations of Section 9, using Eq. (23) in step 3, until once more convergence is obtained.

In small problems, this scheme is slightly slower than using Eq. (23) from the start, since the total number of iterations is somewhat increased. In large problems, this scheme can be several fold faster, since generally only one or two applications of Eq. (23) are required.

12. Auxiliary processing unit

The VM/370 System can be run on a so-called auxiliary processor (AP) configuration containing dual instruction processing units. One of these, the CPU, can handle all instructions, including I/O. The other one, the APU, cannot execute I/O instructions. Any program can be dispatched on either unit, but, if an I/O instruction is encountered on the APU, the task must be suspended and scheduled to run on the CPU. In addition, I/O interrupts occur only on the CPU, so that the interrupt handler runs only there. Thus, a certain fraction, say F, of all the instructions can only be executed on the CPU. The value of F has been measured as about 10 percent of total supervisor time. Furthermore, it has been determined that about 50 percent more supervisor time is incurred by a transaction running on an AP configuration than on a uniprocessor (UP) configuration, due to additional dispatching and scheduling overhead, lock spinning, and signaling between processors. Thus, measured CPU supervisor time has to be inflated by 50 percent before a workload measured on a UP can be modeled on an AP.

Table 2 Comparison of asymptotic and finite multiprogrammed set models, with $t_i = 6, 3, 1$ and $\theta_i = 2, 8, 5$.

No. of users in multiprogrammed set			*Time in multiprogrammed set Finite model [Eq. (23)] Asymptotic model [Eq. (29)]*		
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$	$T_{1,3}$	$T_{2,3}$	$T_{3,3}$
1	2	3	19.19	18.51	8.71
			20.87	17.43	8.14
2	4	6	53.28	34.57	14.03
			54.98	34.49	13.83
4	8	12	125.31	70.02	25.77
			126.12	70.06	25.69
8	16	24	269.32	141.82	49.66
			269.72	141.86	49.62
3	2	1	28.46	22.47	10.00
			29.17	21.59	9.53
6	4	2	63.48	39.39	15.60
			64.04	39.02	15.34
12	8	4	135.05	74.84	27.36
			135.39	74.69	27.23
24	16	8	278.86	146.58	51.24
			279.04	146.52	51.17

Since AP configurations are generally designed to serve heavy loads with many users, it was decided that only the asymptotic model need be adapted. The only factor that needs to be adjusted is $f_1(n_1)$, the processing unit's service rate. Assume that $n \geq 2$. The probability that all n_1 programs require the CPU is F^{n_1}. Hence, $f_1(n_1) = 1$ with probability F^{n_1}, and $f_1(n_1) = 2$ with probability $1 - F^{n_1}$. Thus, on the average,

$$f_1(n_1) = F^{n_1} + 2(1 - F^{n_1}) = 2 - F^{n_1}. \tag{30}$$

Remembering that $n_1 = Nx_1$, we modify (25) to

$$x_1^{(v+1)} = x_1^{(v)} \sum_i \frac{a_i t_i}{\dfrac{t_i x_1^{(v)}}{2 - F^{Nx_1^{(v)}}} + \beta_i}. \tag{31}$$

Similarly, (29) should be modified to

$$T_{i,3} = \theta_i + \max\left(t_i, \frac{Nx_1 t_i}{2 - F^{Nx_1}}\right). \tag{32}$$

Table 3 Validation of model on real workloads. Note that predictions differ somewhat from those presented in [5] due to changes in the model.

CPU model	*Average logged users*	*Percent CPU utilization*		*Percent virtual CPU time*		*Average trivial response* (seconds)		*Average nontrivial response* (seconds)	
		Measured	*Predicted*	*Measured*	*Predicted*	*Measured*	*Predicted*	*Measured*	*Predicted*
135	4	17.1	17.2	5.3	5.2	0.7	1.0	19.0	24.1
145	8	84.0	84.8	42.5	42.9	0.25	0.24	3.9	3.1
145	15	96.6	97.4	40.8	41.4	0.51	0.44	26.6	19.7
155-II	20	22.2	22.2	6.7	6.6	0.05	0.06	1.2	1.1
155-II	23	36.9	35.7	10.7	10.3	0.08	0.11	2.8	3.6
158	37	59.2	55.4	31.5	28.7	0.21	0.26	21.8	18.4
158	46	70.3	69.0	37.8	36.7	0.14	0.12	2.5	1.6
158	24	68.8	71.3	52.2	55.5	0.07	0.09	6.1	5.3
168	72	36.0	35.2	14.5	14.6	0.13	0.11	7.8	6.7
168	117	96.3	99.7	56.0	57.9	0.46[a]	0.41	8.0	9.7
						0.48[a]	0.53	13.9	10.7
						0.55[a]	0.58	19.2	19.2
						0.83[a]	0.73	28.3	26.0

[a]These response times refer to four separate user classes. Classification was based on ratio of trivial to nontrivial transactions.

Table 4 Description of benchmark workloads.

Class name	*Description*	*Number of users in workload with total user number equal to*		
		20	*40*	*60*
TRVU	Generates a trivial transaction after every 10 seconds of think time	8	16	24
MEDU	Generates a nontrivial transaction (FORTRAN compilation) after every 40 seconds of think time	8	15	23
HVYA	Generates a continuous stream of PL/1 compilations, FORTRAN compilations, and assemblies	2	4	6
HVYB	Generates a continuous stream of assemblies	2	5	7

13. Model outputs

Once the model has converged, many performance measures can be computed easily from available quantities. We list some of these below:

- Class i transaction average response time:

$$T_i = T_{i,2} + T_{i,3}.$$

Note that this is the *internal response time*, i.e., the time elapsed from the system scheduler's recognition of the transaction's arrival until the CPU's completion of the transaction processing. It does not include transmission delays between the CPU and the user's terminal. To the model, such delays appear as part of the think time.

- Class i transaction rate, from Eq. (4):

$$\lambda_i = N_{i,3}/T_{i,3}.$$

- Class i CPU utilization:

$$V_i = \lambda_i t_i.$$

- Total CPU utilization:

$$V = \sum_i V_i.$$

(I/O channel and device utilizations are obtained as by-products of the I/O system submodel.)

- Class i transaction wait and service time for each system component (storage, CPU, paging and I/O data sets).

14. Model validation

The model is typically used in the following fashion: The VM/370 monitor facility [9] is turned on to collect data for, say, two hours of peak load time, on a running VM/370 installation. The data are reduced to produce

1. A summary of the system's performance during the measured period;
2. A characterization of the installation's workload.

The latter is input to the model, together with the description of the actual system configuration. The model's output is then compared to the observed system performance. If the two match to the user's satisfaction, the model is considered validated for this installation. It is now possible to vary the model's inputs to reflect changes in workload, configuration, or both, and thus to explore

Table 5 Validation of model on benchmark workload.

Run	*Main storage size* (Kbytes)	*No. of users*	*Percent CPU utilization*		*Percent virtual CPU time*		*Average response*[a] (seconds)	
			Measured	*Predicted*	*Measured*	*Predicted*	*Measured*	*Predicted*
1	512	20	48.4	48.0	35.0	29.7	TRVU 0.33	0.16
							MEDU 32.0	50.5
							HVYA 36.7	46.0
							HVYB 44.0	45 .8
2	1024	20	81.1	95.8	58.1	67.4	0.31	0.21
							15.0	12.2
							15.5	13.7
							16.3	12.3
3[b]	1024	40	84.4	91.9	58.4	59.9	0.38	0.25
							49.1	49.8
							38.8	46.0
							45.8	43.8
4	1024	60	86.7	82.2	50.7	48.4	0.41	0.26
							127.7	122.4
							75.1	83.7
							84.6	84.6
5	2048	40	96.8	100	69.3	73.4	0.25	0.31
							44.8	34.0
							32.6	38.2
							30.0	33.6
6	2048	60	98.5	100	67.5	69.1	0.34	0.44
							78.2	76.5
							61.7	60.8
							57.5	55.1

[a]Response times for HVYA and HVYB are per second of virtual CPU time.
[b]Workload characterization derived from run no. 3.

the system's predicted performance under a wide variety of circumstances. If the match is unsatisfactory, the model can be "tuned" by manipulating some of the more doubtful input parameters, e.g., I/O block lengths, seek patterns, and assignment of disk areas to the various user classes.

Table 3 presents the results of several such validations, which show that the model can reasonably match the observed performance of the measured systems. Of greater interest, however, is the question of how accurately the model can predict the performance of a projected system, rather than that of the actually measured one. Since a true workload measured on an existing system cannot generally be reproduced after the system has been changed, this type of validation requires a controlled benchmark environment. For this purpose, a benchmark stream consisting of four user classes (see Table 4) was run repeatedly on a System/370 Model 158 with varying main storage sizes and total number of users. The data from one of these runs were used to derive the workload characterization, and the model was then invoked to predict the performance for all the runs. Comparisons of predicted and measured performance appear in Table 5. Once more, the accuracy of the predictions is quite adequate for the purposes of configuration and capacity planning.

The predictions are least accurate in the 512-Kbyte case (where K = 1024), primarily due to the difficulty of predicting the paging rate in such a squeezed-storage situation. Remember that in going from 1024 Kbytes to 512 Kbytes the storage available for user programs is actually reduced by a factor of three.

All the above validation results were attained without any attempt to "tune" the model. The results, therefore, indicate what can be achieved by an inexperienced user in a limited amount of time.

15. Conclusions

The model described here has been programmed in APL for use by IBM personnel. A typical case requires 2–20 seconds of System/370 Model 168 time.

The model has proved to be very successful in practice, mostly because of the ease with which input to the model—primarily characterizations of existing workloads—can be obtained [5].

Improved model accuracy can probably be best attained through better modeling of

1. The paging process—particularly better prediction of the relation between paging rate, system configuration, and workload characteristics;

2. Priority dispatching within the multiprogrammed set.

Our model ignores all delays incurred in data transmission between the CPU and remote user terminals. To predict response times actually experienced by users at remote terminals, the model would have to be coupled to a data transmission network model. The same type of iteration described in Section 9 could be used to alternate between the network model and the host VM/370 model.

While some of the details are specific to the VM/370 System, the model should be readily applicable to many other interactive multiprogrammed computer systems. Perhaps the most important feature of VM/370 which makes it easy to model is that the only interference between users in the multiprogrammed set is through contention for physical resources: CPU and I/O paths. Blocking through software locks to prevent simultaneous access to data items is largely absent. However, it should be possible to model software locks so the θ_i computed by the I/O subsystem model reflect these delays, leaving the logic of the rest of the model unchanged. The possibility of modeling other scheduling algorithms would have to be investigated on a case by case basis.

References and note

1. Much of the material in this paper appears in *Computer Performance: Proceedings of the International Symposium on Computer Performance Modeling, Measurement, and Evaluation*, K. M. Chandy and M. Reiser, eds., North-Holland Publishing Co., Amsterdam, 1977, p. 113. The symposium was sponsored by IFIP Working Group 7.3.
2. *IBM Virtual Machine Facility/370, Introduction*, Form No. GC20-1800, IBM Data Processing Division, White Plains, NY, 1972.
3. Y. Bard, "An Analytic Model of CP-67 and VM/370," *Computer Architectures and Networks*, E. Gelenbe and R. Mahl, eds., North-Holland Publishing Co., Amsterdam, 1974, p. 419.
4. J. E. Neilson, "An Analytic Performance Model of a Multiprogrammed Batch-Timeshared Computer," *Proceedings of the International Symposium on Computer Performance Modeling, Measurement and Evaluation*, Cambridge, MA, 1976, p. 59.
5. Y. Bard, "A Characterization of VM/370 Workloads," *Modelling and Performance Evaluation of Computer Systems*, H. Beilner and E. Gelenbe, eds., North-Holland Publishing Co., Amsterdam, 1977, p. 35.
6. P. J. Courtois, "Decomposability, Instabilities, and Saturation in Multiprogrammed Systems," *Commun. ACM* **18,** 371 (1975).
7. C. J. Young, "VM/370 Biased Scheduler," *Technical Report TR 75.0001*, IBM New England Programming Center, Burlington, MA, 1973.
8. *IBM Virtual Machine Facility/370 System Extensions, General Information Manual,* Form No. GC 20-1827, IBM Data Processing Division, White Plains, NY, 1977.
9. *IBM Virtual Machine Facility/370, System Programmer's Guide*, Form No. GC20-1807, IBM Data Processing Division, White Plains, NY, 1976.
10. J. D. C. Little, "A Proof of the Queueing Formula $L = \lambda W$," *Oper. Res.* **9,** 383 (1961).
11. H. Kobayashi and M. Reiser, "On Generalization of Job Routing Behavior in a Queueing Network Model," *Research Report RC-5679*, IBM Thomas J. Watson Research Laboratory, Yorktown Heights, NY, 1975.
12. M. A. Diethelm, "An Empirical Evaluation of Analytic Models for Computer System Performance Prediction," *Computer Performance: Proceedings of the International Symposium on Computer Performance Modeling, Measurement, and Evaluation,* K. M. Chandy and M. Reiser, eds., North-Holland Publishing Co., Amsterdam, 1977, p. 139.
13. M. Reiser and H. Kobayashi, "Queuing Networks with Multiple Closed Chains: Theory and Computational Algorithms," *IBM J. Res. Develop.* **19,** 283 (1975).
14. P. H. Seaman, R. A. Lind, and T. L. Wilson, "On Teleprocessing System Design, Part IV: An Analysis of Auxiliary Storage Activity," *IBM Syst. J.* **5,** 158 (1966).
15. Y. Bard, "Task Queueing in Auxiliary Storage Devices with Rotational Position Sensing," *Technical Report G320-2070*, IBM Cambridge Scientific Center, Cambridge, MA, 1971.
16. N. C. Wilhelm, "A General Model for the Performance of Disk Systems," *J. ACM* **24,** 14 (1977).
17. B. Pittel, "Closed Exponential Network of Queues with Blocking, the Jackson type Stationary Distribution and its Asymptotic Analysis," *Research Report RC 6174*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1976.

Received May 23, 1977; revised April 26, 1978

The author is located at the IBM Cambridge Scientific Center, 545 Technology Square, Cambridge, MA 02139.