An Approach to the GRM Performance Analysis by Asymptotic Approximation

Tohru Nishigaki,* Kenichiro Noguchi** and Kazuhiko Ohmachi*

An approach is presented to analyze the performance of a system with the General Resources Manager (GRM). The GRM is a periodical modifier of resource allocation priorities for throughput/response improvements in multiprogrammed computer systems. Because of the c mplexity, its scheduling effect has so far been investigated by only simulations or measurements.

This paper first considers the system with constant priorities and develops a new approximation technique, termed the Asymptotic Model. Simulations results show the validity of the model for the prediction of response times under priority scheduling disciplines and limited degree of multiprogramming. Secondly, based on the model, the control policy of the GRM is discussed. Furthermore a way to predict the performance under the GRM control policy is introduc-d.

The approach presented reveals very little of GRM's dynamic behavior. For the full analysis of the GRM much work remains to be done. However, some essential properties of the GRM are considered successfully demonstrated by the use of the model.

1. Introduction

The need for the balanced optimization of the two performance objectives—response and throughput—has brought about a new set of schedulers based on a feedback concept [1]–[4]. The OS/VS2 Release 2 System Resources Manager (SRM) [3] was the first explicit attempt to achieve these two goals in various TSS/batch applications. However, the limitation in its effectiveness urged a more general scheduler, the General Resources Manager (GRM) [1],'2]. The SRM controls performance solely by swapping (real memory allocation), and is ineffective in an undercommitted real memory environment. The GRM, on the other hand, controls performance by the allocation of various resources such as real memory, CPU, channel, etc., thus holding effectiveness in diverse environments.

The investigations on the performance for those schedulers were carried out mainly by simulations and/or measurements [1], [2], [4]. They were generally local studies, since the high cost inherent to these evaluation methods prevented more systematic and global studies. The requirement for the investigation of global system behaviors leads us to an analytical approach, especially to queueing network theory. However, the research of this area [8], [9] has shown its inapplicability to our problem. This is because the current queueing network theory does not include priority scheduling disciplines, whereas the control of the GRM/SRM is based on priority adjustment. Theoretical treatments of priority scheduling so far are M/M/1 or M/G/1 or M/M/s setting at most, failing to analyze the interrelation between resources [10], [11]. Even an approximate treatment of a queueing network [15] is too confined for the GRM performance analysis, since it covers solely CPU scheduling. In addition, another difficulty is that the current queueing theory cannot solve the multiple resource holding (secondary resource) problem. The real memory should be treated as the secondary resource, because it is kept allocated to jobs which are waiting for or being served by CPU or channels.

This paper presents an analytical approach to the system performance under the GRM. Since the GRM is the mechanism of periodic priority adjustment, our approach is divided into two stages. The first stage described in **3.** and **4.** concerns the performance during the interval between the two successive priority adjustments. In other words, this stage is the analysis of the system with static (constant) priorities. A new approximation model, including both priorities and real memory as a secondary resource, is developed. The basic idea is to approximate the exact solution by its asymptote. Muntz et al. [5]–[7], [14] indicated the mean response time has an asymptote as a function of customer population size (the number of active terminals). However, their model assumes a single job class and unlimited degree of multiprogramming, and neither priorities nor real memory is included. Our model, being in the extension of their approach, is based on two assumptions. One is the neglect of queueing delays when the resource can never be utilized to capacity. And once it can, the resource is assumed always utilized to capacity. The other is the assumption of preemptiveness, which implies that a job is never influenced by lower priority ones. These are the assumptions to obtain asymptotes in multiple job class environments. And asymptotic

*Systems Development Laboratory, Hitachi, Ltd.
**Software Works, Hitachi, Ltd.

solutions enable us to have a rough estimate of the global system behavior under complex priority scheduling. Detailed simulations are carried out in **4.** to validate our model.

The second stage described in **5.** concerns the long range performance under periodic priority adjustment. This stage is much more difficult than the first one. It is because the strict argument must include the transient effects such as the possible influence by the lower priority jobs immediately after the priority adjustment. The analysis of these transient effects is a further research concern, and this paper confines itself to giving some analytical insights into the performance under the GRM control. By using the first stage model, the effectiveness of the GRM control policy is discussed. Furthermore, one way to predict the performance under the policy is presented.

2. The System under the GRM Control

The system is represented by the finite population model with limited degree of multiprogramming. In Fig. 1 each terminal initiates a transaction at every think time expiration. In other words, "think time" in this model is the time interval between the completion of a transaction and the arrival of the following one. A resource service requester is called a "transaction" which corresponds to a batch job or TSS command. Batch jobs in this figure are considered to be initiated by the terminals for which think times are set at zero. (The queueing delay of job-scheduling is not covered in this paper.)

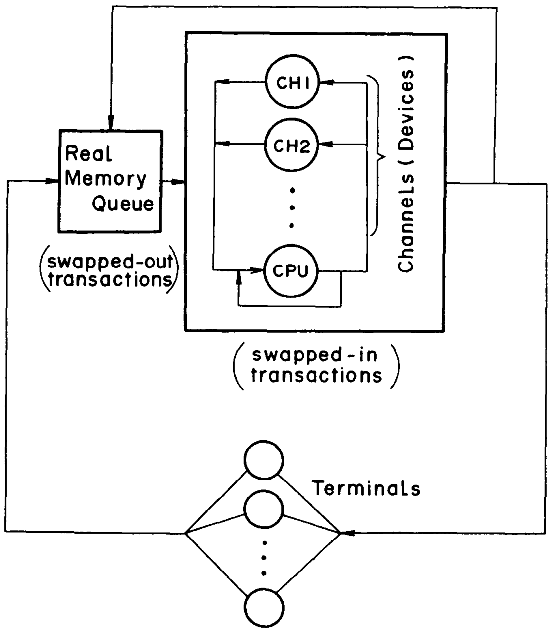


Fig. 1 The system model—finite population model with limited degree of multiprogramming.

Since the paging load varies with the multiprogramming degree, some mechanism for the multiprogramming degree adjustment is indispensable to avoid thrashing [12], [13]. This problem is out of this paper's concern, and working-sets are assumed rightly chosen to always realize the light/moderate paging load. The maximum multiprogramming degree s is obtained as V/w, where V and w represent the total real memory amount and the mean working-set size respectively. For simplicity, V is assumed a multiple of w. In case the number of non-dormant (not in the course of "thinking") terminals exceeds s, the excess transactions are queued for the real memory allocation and wait to be swapped in.

The GRM periodically changes the priority for each resource allocation for each transaction, while aiming at balanced improvements in throughput and response. Its control policy includes that of the SRM which concerns only real memory allocation [3]. Moreover, the GRM also covers traditional time-slicing control and dynamic dispatching control. A detailed description is given in [1], [2] and here only an outline is presented.

The GRM has two discrete and potentially conflicting control objectives. The first one is throughput improvement—the better use of resources. Let u_j be the utilization of resource j and q_j be the lower limit of the acceptable range. Resource j is defined to be either "busy" if $u_j \geq q_j$, or "idle" if $u_j < q_j$. The control objective is given by

$$u_j \geq q_j \tag{1}$$

where j refers to CPU or a channel, since only CPU and channels need to be busy for throughput maximization.

The second objective is response improvement. It is accomplished by an adequate resource service distribution. The resource service rate, which is the resource service amount being supplied to a transaction per second, is related to a service objective function called the Performance Objective [1]–[3]. This monotonously decreasing function represents the relative importance of the transaction, and specifies its ideal service rate as a function of the workload level. Let r_i be the transaction i's service rate and g be the Performance Objective associated with i. The Normalized Workload Level (NWL) of i is given by

$$\mathrm{NWL}_i = g^{-1}(r_i) \tag{2}$$

where g^{-1} represents the inverse function of g (see Fig. 2). The control objective is the reduction of the NWL's variance, namely

$$\mathrm{NWL}_i \rightarrow \overline{\mathrm{NWL}} \quad (i=1, 2, \cdots) \tag{3}$$

where $\overline{\mathrm{NWL}}$ represents the mean of NWL. Note that eq. (3) assures a relatively high service rate (short response time) for an emergent transaction with which a high Performance Objective is associated [1]–[3].

The GRM control policy is an integrated priority assignment scheme that is utilized to attain eqs. (1) and

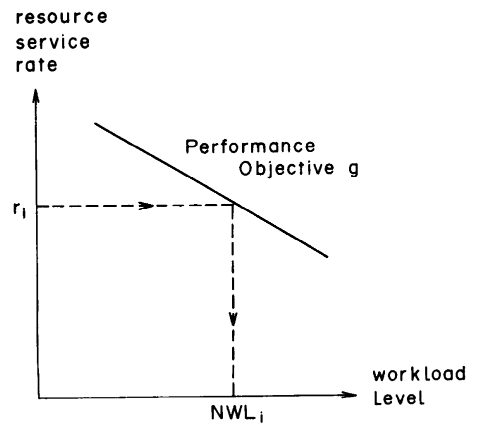


Fig. 2 The computation of Normalized Workload Level of transaction *i*.

(3). In this policy, the concept of predecessor/successor is introduced. If some transaction has to be allocated resource J1 in order to request the service of resource J2, J1 is called J2's predecessor and J2 is called J1's successor. (Naturally a successor of J2 is also J1's successor.) For example, real memory is a predecessor of CPU and channels, and CPU is a predecessor as well as a successor of channels. Assuming that J1 is "busy" and J2 is "idle", the utilization of J2 can be improved by giving precedence to J2's heavy user (the transaction which uses J2 by a large amount) in the allocation of J1. In addition, the variance of NWL can also be controlled by adjusting the allocation priorities for J1. This is because a "busy" resource is a key resource in service distribution.

The GRM control policy is summarized as follows. (1) Increase the priorities of the idle resource's heavy users, for allocation of such resources as are busy and predecessors of the idle resource. (2) Increase the priorities of those transactions which have relatively large NWL and decrease the priorities of those transactions which have relatively small NWL, for allocation of busy resources.

3. The Asymptotic Model for Priority Scheduling

3.1 Single Class Model

First the model for a single transaction class without priorities is discussed as an introduction. Let T, R, z and s be the mean response time, mean resident time, mean think time and maximum multiprogramming degree respectively. R represents the mean time when real memory is allocated to a transaction. In other words, R is T minus the mean queueing delay for real memory allocation. Since CPU and channels are treated equally, they are all called "processors" hereafter. The mean use time of processor j by a transaction is denoted as d_j. The lower bound of mean response time is given by

$$R_0=\sum_j d_j. \tag{4}$$

Note that R_0 corresponds to the mean response time without queueing delays.

The utilization of processor j u_j and that of real memory u_m satisfy

$$\begin{cases} u_j=Nd_j/(T+z) & (j=1,2,\cdots) \\ u_m=N(R/s)/(T+z) \end{cases} \tag{5}$$

where N is the customer population size (the number of active terminals). Note that eq. (5) holds for any inter-arrival/service time distribution. Letting $\max_j d_j$ be d_x, eq. (5) yields

$$\max_j u_j = u_x \tag{6}$$

for any value of N.

The first assumption for our approximation model is as follows:

ASP1. The queueing delay for a resource service is neglected when the resource can never be utilized to capacity. And when it can, the resource is assumed always utilized to capacity.

This is the assumption to obtain asymptotic solutions. It should be noted that asymptotic solutions correspond to the "minimal" queueing delays for any inter-arrival/service time distribution. Apparently, actual queueing delays can be significant even when the resource is not utilized to capacity. The difference between exact and asymptotic solution depends upon inter-arrival/service time distribution. Since the distributions in a queueing network with priority scheduling disciplines are generally not known [8], [9], [15], our approach is to approximate queueing delays by their asymptotes. As a general rule, this approach tends to underestimate the actual response time.

In case s is infinite, ASP1 leads to the following simple approximation which is widely known [5], [6].

$$T=\begin{cases} R_0 & : 1\le N<(R_0+z)/d_x \\ d_xN-z & : \text{otherwise.} \end{cases} \tag{7}$$

The approximation with finite value of s is obtained by using ASP1 and eq. (5) as follows:

$$\begin{cases} T=d_xN-z\text{: for } N \text{ yielding } u_x=1 \\ R=R_0\text{: otherwise.} \end{cases} \tag{8}$$

$$T=\begin{cases} (R/s)N-z\text{: for } N \text{ yielding } u_m=1 \\ R \quad : \text{otherwise.} \end{cases} \tag{9}$$

The second equations in eq. (8) and eq. (9) each indicate the neglect of queueing delays when the population size is too small to attain the full utilization of the resource. Thus we obtain from eqs. (8) and (9),

$$T=\begin{cases} d_xN-z & : \text{for } N \text{ yielding } u_x=1 \\ (R_0/s)N-z & : \text{for } N \text{ yielding } u_x<1 \ \&\ u_m=1 \\ R_0 & : \text{otherwise.} \end{cases} \tag{10}$$

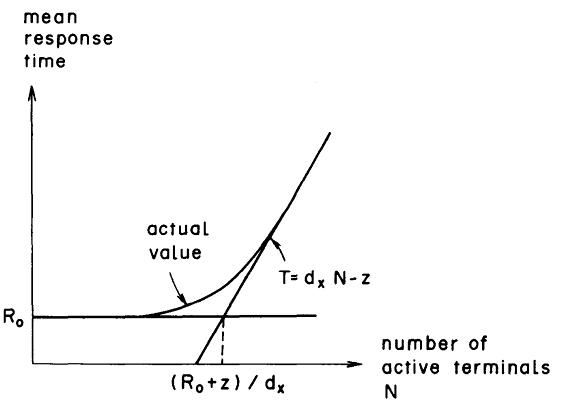


Fig. 3 The asymptotes of mean response time T with unlimited degree of multiprogramming.

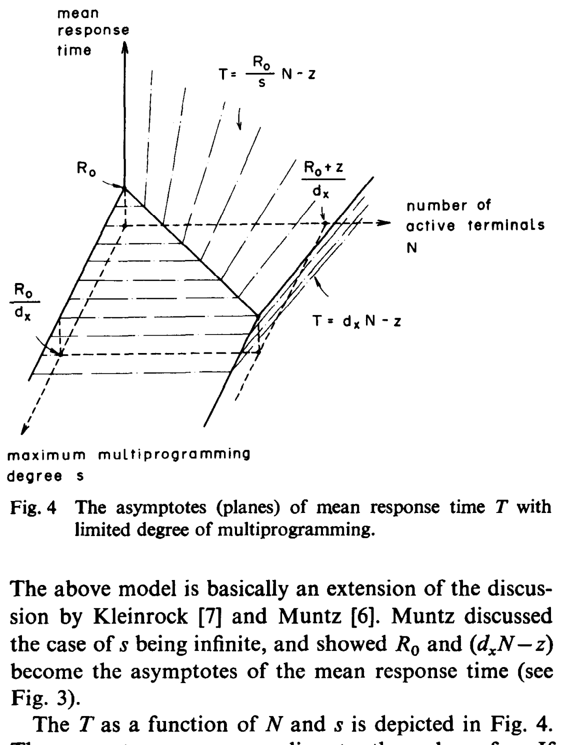


Fig. 4 The asymptotes (planes) of mean response time T with limited degree of multiprogramming.

The above model is basically an extension of the discussion by Kleinrock [7] and Muntz [6]. Muntz discussed the case of s being infinite, and showed R_0 and (d_xN-z) become the asymptotes of the mean response time (see Fig. 3).

The T as a function of N and s is depicted in Fig. 4. There are two cases according to the value of s. If $1 \leq s < R_0/d_x$, the system becomes "memory neck" but never both "processor x neck" and "memory neck". (Hereafter we call the system is "processor x neck" if $u_x = 1$, and "memory neck" if $u_m = 1$.) This is because eqs. (5), (6), (8), (9) give eq. (11) in the memory neck situation $(N \geq (R_0+z)s/R_0)$.

$$u_x = sd_x/R_0 < 1. \tag{11}$$

On the other hand, if $s \geq R_0/d_x$, it becomes first "processor x neck" and then both "processor x neck" and "memory neck" as N increases. This can be explained by the next equation, which is given by eqs. (5), (6), (8), (9) in the processor neck (and not memory neck) situation $(\{s+(z/d_x)\} > N \geq (R_0+z)/d_x)$.

$$u_m = \{N-(z/d_x)\}/s. \tag{12}$$

Eq. (12) tends to 1 as N approaches $\{s+(z/d_x)\}$. Obviously T depicted in Fig. 4 coincides with the Muntz's result when s is infinite.

3.2 Multiple Class Model

The system is assumed to have the static (constant) priorities. Any variable related to class i transaction is indicated by an index i such as T_i, R_i, d_{ij}, R_{i0}, z_i, s_i, N_i, etc. Using eq. (5) we obtain

$$\begin{cases} u_j = \sum_i N_i d_{ij}/(T_i+z_i) & (j=1,2,\cdots) \\ u_m = \sum_i N_i(R_i/s_i)/(T_i+z_i). \end{cases} \tag{13}$$

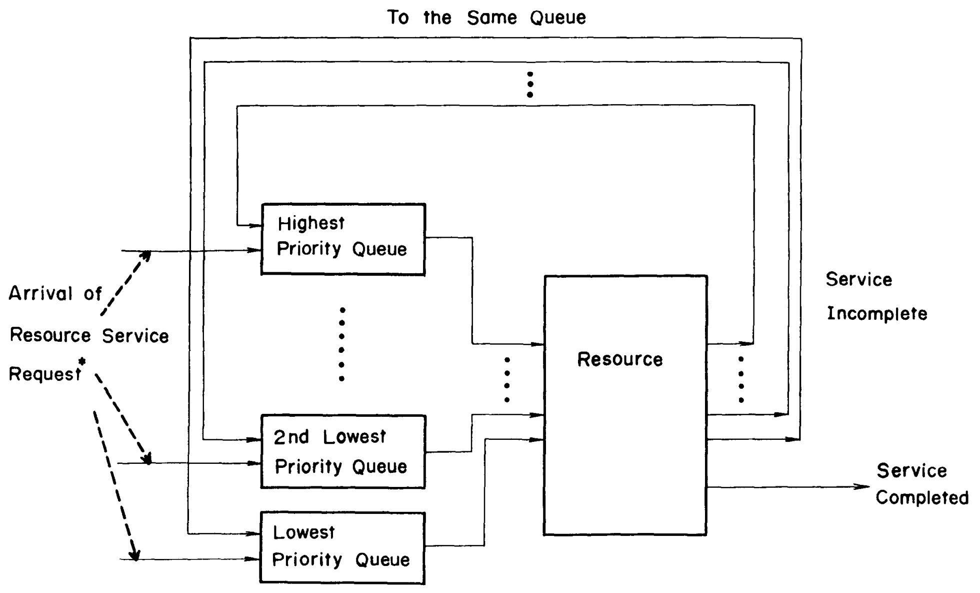
Another assumption is introduced in addition to ASP1.

ASP2. Preemptive resume priority scheme is assumed. Therefore a transaction is never affected by lower priority transactions. The lower priority transactions will be supplied with only the remainder of the resource service.

Channels in an actual system are nonpreemptive. And even real memory can be sometimes nonpreemptive, since a transaction in the course of I/O operation cannot be swapped out until its completion. Therefore ASP2 makes the model underestimate the high priority transaction's response time, and overestimate the low priority transaction's response time.

Unlike the single class case, the range of $N(=\sum_j N_i)$ to be analyzed is limited, so as to ensure a finite value of T_i for any transaction class. In other words, the number of high priority customers is assumed small enough to utilize none of the resources to capacity. This is because the low priority transaction's response time becomes infinite when some resource is completely occupied by the high priority transactions. This, as combined with ASP1 and ASP2, means that the queueing delay at a resource is neglected except for its lowest priority transactions. (The limitation in N does not affect the generality of the argument. If T_i is infinite, other classes' response times can be obtained by simply neglecting class i in the analysis.)

The service discipline at a resource in the model is as follows (see Fig. 5): (1) There are as many priority queues as priority levels. A transaction in a queue is served only when the higher priority queues are all empty. (2) No particular discipline is assumed for the service to the transactions in a given queue (except for the neck resource's lowest priority queue). It may be First Come First Served, Processor Sharing, etc. This is because the asymptotic solution is the same for any service discipline, although the actual queueing delays might be affected by it. As for the lowest priority queue of a neck resource, it is assumed that the service is distributed among the lowest priority transactions on a



To the Same Queue
Highest Priority Queue
2nd Lowest Priority Queue
Lowest Priority Queue
Resource
Arrival of Resource Service Request*
Service Incomplete
Service Completed
* PRIORITIES ARE DETERMINED PERIODICALLY TO ACHIEVE eqs. (1) and (3).

Fig. 5. The service discipline for the multiple class model.

"fair share" basis [17]. That is, the average use amount of the neck resource is the same for any active terminal which initiates the lowest priority transactions.

The equations to obtain asymptotic solutions are presented below. First the model of infinite value of s_i is introduced. Let y_j be the set of processor j's lowest priority transaction classes, and H be the set of processors of which the utilizations are 1. Note that all transaction classes become the member of y_j, unless processor j is under priority scheduling. In case H is empty, $T_i = R_{i0}$ for any i. Otherwise they are given by next equations.

$$\sum_i \{(1-e_i)N_i d_{ij}/(R_{i0}+z_i)+e_i N_i d_{ij}/(T_i+z_i)\}=1 \qquad (j \in H) \quad (14)$$

$$d_{i_1 j}/(T_{i_1}+z_{i_1})=d_{i_2 j}/(T_{i_2}+z_{i_2})=\cdots \qquad (i_1, i_2, \cdots \in y_j; j \in H) \quad (15)$$

where

$$e_i=\begin{cases}1: \text{ if } \exists k \in H \text{ such that } i \in y_k \\ 0: \text{ otherwise.}\end{cases} \quad (16)$$

Eq. (14) is obtained by substituting T_i by R_{i0} in eq. (13), for such i as joins none of $y_k (k \in H)$. The fair share service discipline [17] is reflected in eq. (15). In other words, eq. (15) defines "the equality of priority" in this model. The number of unknown variables T_i is $|\bigcup_{j \in H} y_j|$. (Hereafter $|\ \ |$ denotes the number of members in a set, and $\bigcup$ denotes the union of sets.) The numbers of eqs. (14) and (15) are $|H|$ and $\sum_{j \in H}(|y_j|-1)$ respectively. Therefore we can obtain the solution if

$$|\bigcup_{j \in H} y_j| = \sum_{j \in H} |y_j|. \quad (17)$$

This mutual exclusiveness of y_j can be interpreted that a transaction in this model never waits at two or more processors. The T_i's given by eqs. (14)–(16) must satisfy

$$\begin{cases}0 \le u_j \le 1 & (j=1, 2, \cdots) \\ R_{i0} \le T_i & (i=1, 2, \cdots).\end{cases} \quad (18)$$

In other words, H is determined by eqs. (13)–(16), (18) for each set of values of N_i. It is noteworthy that eqs. (14) and (15) can be reduced to independent linear equations if eq. (17) holds. The reduction is done by equating each of eq. (15) with a new variable of j and substituting T_i's by it.

The finite values of s_i might cause the memory neck situation. In case they do not, T_i's are given by eqs. (14)–(16). Therefore we assume $u_m = 1$ hereafter, and the set of transaction classes with the lowest real memory allocation priority is denoted as y_m.

The next equations are those for finite s_i, given as an extention of eqs. (14)–(16).

$$\sum_i [(1-e_i)(1-e_{im})N_i d_{ij}/(R_{i0}+z_i) + \{1-(1-e_i)(1-e_{im})\}N_i d_{ij}/(T_i+z_i)]=1 \qquad (j \in H) \quad (19)$$

$$d_{i_1 j}/(T_{i_1}+z_{i_1})=d_{i_2 j}/(T_{i_2}+z_{i_2})=\cdots \qquad (i_1, i_2, \cdots \in y_j; j \in H) \quad (20)$$

$$\sum_i [(1-e_i)(1-e_{im})N_i(R_{i0}/s_i)/(R_{i0}+z_i)$$
$$+\{e_i(1-e_{im})\}N_i(T_i/s_i)/(T_i+z_i)$$
$$+\{(1-e_i)e_{im}\}N_i(R_{:0}/s_i)/(T_i+z_i)$$
$$+(e_ie_{im})N_i(R_i/s_i)/(T_i+z_i)]=1 \qquad (21)$$

$$\{(1-e_{i_1})(R_{i_10}/s_{i_1})+e_{i_1}(R_{i1}/s_{i1})\}/(T_{i_1}+z_{i_1})$$
$$=\{(1-e_{i_2})(R_{i_20}/s_{i_2})+e_{i_2}(R_{i_2}/s_{i_2})\}/(T_{i_2}+z_{i_2})=\cdots$$
$$(i_1,\ i_2,\cdots\in y_m) \qquad (22)$$

where

$$e_{im}=\begin{cases}1:\ \text{if } i\in y_m\\ 0:\ \text{otherwise.}\end{cases} \qquad (23)$$

Eqs. (19), (20) are the same as eqs. (14), (15), except that T_i includes the queueing delay of real memory allocation for i in y_m. The equation $u_m=1$ corresponds to eq. (21), where the relation $T_i=R_i$ is assumed from ASP1 and ASP2 unless i belongs to y_m. The fair share service of real memory among its lowest priority transactions is reflected in eq. (22). The numbers of unknown variables T_i and R_i are $|(\bigcup_{j\in H} y_j)\bigcup y_m|$ and $|(\bigcup_{j\in H} y_j)\bigcap y_m|$ respectively, and the sum of them is $\{|\bigcup_{j\in H} y_j|+|y_m|\}$. (Here $\bigcap$ denotes the intersection of sets.) On the other hand, the numbers of eqs. (19)–(22) are $|H|$, $\sum_{j\in H}(|y_j|-1)$, 1, $(|y_m|-1)$ respectively, and the sum of them is $(|y_m|+\sum_{j\in H}|y_j|)$. Therefore the above equations can be solved if eq. (17) holds. In addition, eq. (17) eables us to reduce eqs. (19)–(22) to independent linear equations. The reduction procedure is, essentially, to equate each of eq. (20) and eq. (22) with a new variable and substitute T_i's and R_i's by it. The solution has to satisfy the next conditions as well as eq. (18).

$$R_{i0}\le R_i\le T_i \quad (i=1,2,\cdots). \qquad (24)$$

Note that y_m and y_j are not necessarily mutually exclusive; i.e., a transaction may wait at real memory in addition to a processor. This has already been exemplified in 3.1.

4. Example

A simple example is studied and compared with simulation results. In the example, N terminals are divided equally into two, corresponding to transaction class A and B ($N_A=N_B=N/2$). Since all related variables such as R_{i0}, d_{ij}, s_i, z_i, etc. are assumed the same in the two classes, the indices A and B are omitted hereafter. A transaction utilizes two channels CH1 and CH2 (ee Fig. 1.), each of which a disk is assumed to be connected to. CH1 and CH2 might be interpreted to execute file I/O and paging I/O operations respectively. However, a channel in this example is assumed never to become a neck resource; that is, d_x is always d_{cpu}. This assumption is considered reasonable, because light/moderate channel loading is possible by I/O buffer size modification. Moreover the working-set size is assumed large enough to prevent thrashing.

Two priority scheduling disciplines (1) P(A, CPU)>P(B, CPU) & P(A, MEM)>P(B, MEM) (2) P(A, CPU)>P(B, CPU) & P(A, MEM)<P(B, MEM), are investigated, where P(i, CPU) and P(i, MEM) represent the priority of class i transaction for the allocation of CPU and real memory respectively. The next results are obtained by using eqs. (19)–(23).

(1) P(A, CPU)>P(B, CPU) & P(A, MEM)>P(B, MEM)

(a) $s<R_0/d_{\text{cpu}}$ (memory neck)

$1\le N<s(R_0+z)/R_0$:

$T_A=T_B=R_0$

$s(R_0+z)/R_0\le N<2s(R_0+z)/R_0$:

$$\begin{cases}T_A=R_0\\ T_B=[R_0(R_0+z)N/\{2s(R_0+z)-R_0N\}]-z\end{cases}$$

(b) $R_0/d_{\text{cpu}}\le s$ (CPU neck, CPU & memory neck)

$1\le N<(R_0+z)/d_{\text{cpu}}$:

$T_A=T_B=R_0$

$(R_0+z)/d_{\text{cpu}}\le N<2(R_0+z)/d_{\text{cpu}}$:

$$\begin{cases}T_A=R_0\\ T_B=[d_{\text{cpu}}(R_0+z)N/\{2(R_0+z)-d_{\text{cpu}}N\}]-z\end{cases}$$

(2) P(A, CPU)>P(B, CPU) & P(A, MEM)<P(B, MEM)

(a) $s<R_0/d_{\text{cpu}}$ (memory neck)

the same as (1) (a), by exchanging A for B

(b) $R_0/d_{\text{cpu}}\le s<(2R_0+z)/d_{\text{cpu}}$ (CPU neck, CPU & memory neck)

$1\le N<(R_0+z)/d_{\text{cpu}}$:

$T_A=T_B=R_0$

$(R_0+z)/d_{\text{cpu}}\le N<s+(z/d_{\text{cpu}})$:

$$\begin{cases}T_A=R_0\\ T_B=[d_{\text{cpu}}(R_0+z)N/\{2(R_0+z)-d_{\text{cpu}}N\}]-z\end{cases}$$

$s+(z/d_{\text{cpu}})\le N<2\{s+(z/d_{\text{cpu}})\}$:

$$\begin{cases}T_A=[d_{\text{cpu}}(R_0+z)N/\{2(z+sd_{\text{cpu}})-d_{\text{cpu}}N\}]-z\\ T_B=[d_{\text{cpu}}(R_0+z)N/\{d_{\text{cpu}}N-2(sd_{\text{cpu}}-R_0)\}]-z\end{cases}$$

(c) $(2R_0+z)/d_{\text{cpu}}\le s$ (CPU neck)

the same as (1) (b)

Detailed simulations were carried out to investigate the errors of the above results. The simulation model was developed in CSS (Computer System Simulator), which is an event-driven type simulation language [18]. To imitate the actual system as closely as possible, I/O operations were simulated nonpreemptive, based on the seek time distribution of HITAC H-8589-1 disk. (Refer to [16] for its seek time curve. Random access was assumed.) On the other hand, since the distributions of think time and CPU service time are generally not known, we simulated two distributions—exponential and deterministic. If the actual distribution is Erlangian, the result should be between the above two. The values used in the simulations were: $d_{\text{cpu}}=0.285$ sec, $d_{ch1}=0.086$ sec, $d_{ch2}=0.129$ sec, $R_0=0.5$ sec, $z=20.0$ sec, $s=1\sim20$.

The scheduling effect of discipline (1) and (2) are shown in Fig. 6 and Fig. 7 respectively. The memory neck situation (1) (a), the CPU or CPU & memory neck situation (1) (b), are each exemplified by $s=1$ and $s=10$ (or 20) in Fig. 6. Naturally $T_A\le T_B$ for any N. In the CPU

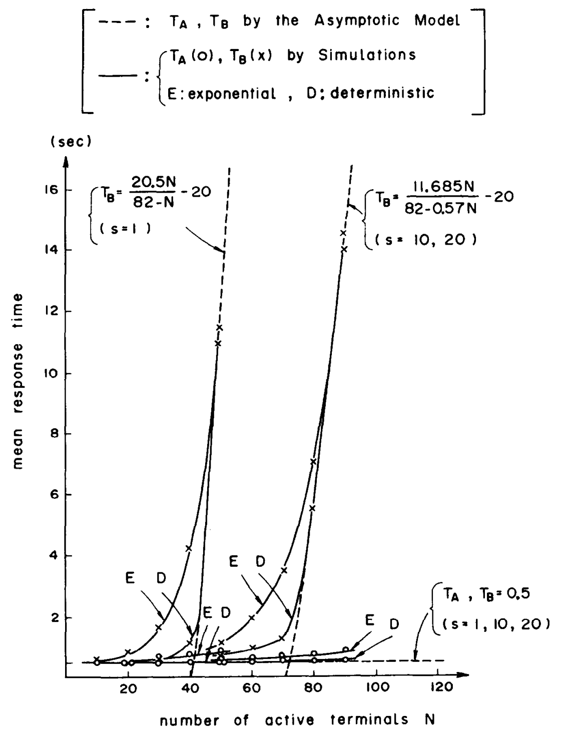


Fig. 6 The validation of the Asymptotic Model by simulations in the case of P(A, CPU) > P(B, CPU) & P(A, MEM) > P(B, MEM).

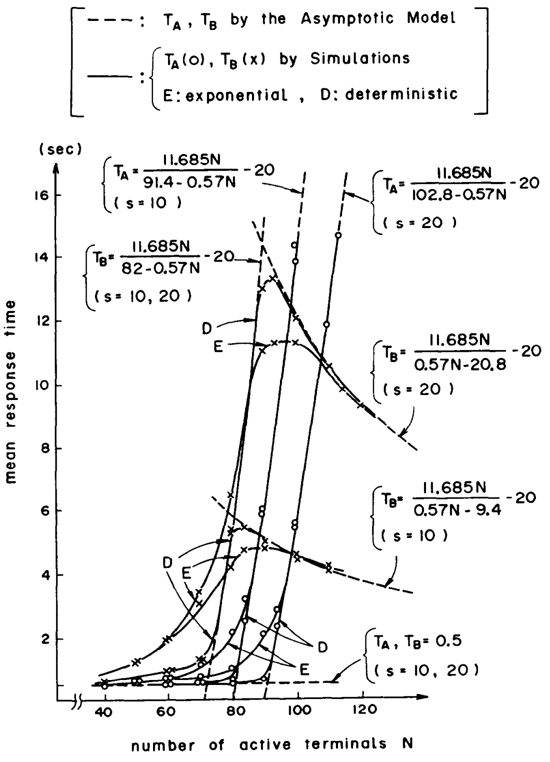


Fig. 7 The validation of the Asymptotic Model by simulations in the case of P(A, CPU) > P(B, CPU) & P(A, MEM) < P(B, MEM).

neck situation, T_B of (1) (b) is the same whether or not real memory is a neck. Therefore no difference is noticed between $s=10$ and 20. The error of T_B becomes significant at the critical point of N, above which a resource can be utilized to capacity. And this is especially true for exponential service.

The examples of (2) (b) are shown in Fig. 7 ($s=10$, 20). The case of (2) (a) and (2) (c) are omitted for simplicity. An interesting behavior of T_B is indicated in Fig. 7—decreasing as a function of N in the CPU & memory neck situation. The CPU neck situation causes queueing delays in T_B, but once it becomes CPU & memory neck, real memory begins to be occupied by class B transactions, resulting in increased T_A owing to the real memory queueing delay. Consequently, the CPU queueing delay decreases, and notable reduction in T_B is achieved. In Fig. 7, the overestimation of T_B can be explained by the overestimated CPU utilization by class A transactions as accompanied by the underestimation of T_A.

The results are summarized as: (1) The error of the Asymptotic Model is negligible when the load is either heavy or light and significant when it is moderate, as is easily seen from ASP1. The increased variance in service time distribution causes greater errors. (2) Even with considerable errors, the Asymptotic Model is useful to investigate the rough behavior of the system with various priority scheduling disciplines.

5. Considerations on the GRM Control

The analysis of the system performance under the GRM dynamic priority adjustment is not possible by the simple combination of static priority cases. It is because the changes in priorities cause transient effects. The argument of these effects have not been obtained, and this paper only gives some analytical insights into the GRM performance control.

There are two objectives, eqs. (1) and (3), in the GRM, and each corresponds to control policy (1) and (2). First we examine eq. (1). Assume that processor j's utilization is very low ($u_j \ll q_j$). Let i be the transaction which has large d_{ij}. (It might make the following argument easier to understand to consider a transaction class is composed of only one transaction; since each transaction in a class is not identified in **3.**) The control policy (1) mentioned in **2.** urges the increase in i's priority for the allocation of "currently neck* and j's predecessor" resources. It is shown in **3.** that this change will decrease T_i. Consequently, u_j given by eq. (13) tends to increase, resulting in the achievement of the objective eq. (1).

Secondly the objective eq. (3) is examined. Let C_i be the total resource service amount to complete transaction

*A "busy" resource can be approximated by a "neck" resource.

i, and r_i be i's resource service rate. Naturally,

$$T_i = C_i / r_i. \tag{25}$$

Assume the NWL of i is too large ($\mathrm{NWL}_i \gg \overline{\mathrm{NWL}}$). The control policy (2) urges the increase in i's priorities for the allocation of "neck" resources. It is shown in 3. that this adjustment will decrease T_i. From eq. (25) and eq. (2), we can see NWL_i is expected to decrease, since NWL_i is monotonously decreasing as a function or r_i. As the similar argument can be applied to the case of $\mathrm{NWL}_i \ll \overline{\mathrm{NWL}}$, it is considered that the objective eq. (3) is achieved. Note that the essential feature of the GRM is to control performance by the priority adjustment at the neck resources. The SRM, on the other hand, is not expected to achieve eqs. (1), (3) in a non-memory neck situation, since it controls performance only by swapping [1]–[3].

The two objectives might conflict with each other, making the performance prediction virtually impossible. This paper concentrates on the case where the response is of supreme importance. In other words, it is such a situation as only eq. (3) is pursued and eq. (1) is neglected. Therefore we can make the following assumption.

$$\mathrm{NWL}_i \cong \overline{\mathrm{NWL}} \quad (i = 1, 2, \cdots). \tag{26}$$

Even with eq. (26), however, the response time is hardly predictable if a transaction is associated with a Performance Objective of complex shape. Two simple types of Performance Objective are studied below: (1) Performance Objective yielding static priorities (2) Performance Objective yielding constant ratio of r_i. The examples of type (1) and type (2) are shown in Fig. 8. Here g_A and g_B each represent the Performance Objective associated with transaction A and B respectively. Since r_i never exceeds C_i/R_{i0} ($i = A, B$), the next relation holds for the type (1) example.

$$L_1 \leq \mathrm{NWL}_B \leq L_2 \leq \mathrm{NWL}_A \leq L_3. \tag{27}$$

This means that A's priority is always kept higher than that of B for any resource allocation. In the type (2) example, on the other hand, the ratio of r_A to r_B is always h_A/h_B regardless of fluctuations in $\overline{\mathrm{NWL}}$.

From a practical viewpoint, these two types are considered sufficient to cope with various user requirements. The former was already discussed in 3., and here we focus on the latter. Given the set of Performance Objectives which yield

$$r_1 : r_2 : r_3 : \cdots = h_1 : h_2 : h_3 : \cdots \tag{28}$$

we obtain from eq. (25),

$$C_1/(T_1 h_1) = C_2/(T_2 h_2) = C_3/(T_3 h_3) = \cdots. \tag{29}$$

The utilizations of resources are given by eq. (13). Therefore if both $\sum_i N_i d_{ij}/(R_{i0} + z_i)$ ($j = 1, 2, \cdots$) and $\sum_i N_i (R_{i0}/s_i)/(R_{i0} + z_i)$ are less than 1, T_i coincides with R_{i0} ($i = 1, 2, \cdots$). Namely eq. (28) does not hold in a system without neck resources. Otherwise some processor and/or real memory become the neck resources. If the system is processor x neck, T_i's are obtained by using eq. (29) and

$$1 = \sum_i N_i d_{ix}/(T_i + z_i). \tag{30}$$

On the other hand, if the neck resource is solely real memory, eq. (29) and

$$1 = \sum_i N_i (R_{i0}/s_i)/(T_i + z_i) \tag{31}$$

give the solution.

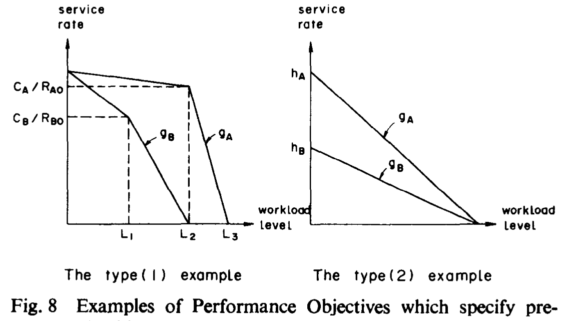


Fig. 8 Examples of Performance Objectives which specify predictable response times.

6. Conclusion

The Asymptotic Model developed gives approximate throughput/response of the system with priority scheduling disciplines and a limited degree of multiprogramming. The detailed simulations ensured that this model is useful to predict the global system behavior.

The analysis based on the Asymptotic Model makes it possible to examine the control policy of the GRM and showed its relevancy to achive the two objectives—improved throughput and response. In addition, the analysis presents a way to predict the response times under the GRM control, for two types of service objective functions.

This approach is expected to give analytical insight into the system performance with complex scheduling disciplines, which has so far been investigated solely by simulations and measurements. However, it should be noted that this approach is only the first step toward the GRM performance analysis. The estimation of approximation errors is to be given more consideration. In addition, the dynamic system behavior under the GRM control is still unknown. More precise arguments, including transient effects caused by dynamic priority changes, would be a further research concern.

Acknowledgements

The authors would like to thank Prof. S. Ohsuga of the University of Tokyo and Prof. H. Kameda of the University of Electro-Communications for their helpful discussions and comments. The authors are also indebted to Dr. T. Miura of Hitachi Systems Development

Laboratory and Mr. Y. Hattori of Hitachi Software Works for the opportunity of study.

References

1. Nishigaki, T. The General Resources Manager based on Feedback Concept in Multiprogrammed Computer Systems, *Joho-Shori*, **19**, 11 (1978) 1026–1033.
2. Nishigaki, T., Ikeda, C., Ohmachi, K. and Noguchi, K. An Experiment on the General Resources Manager in Multi-programmed Computer Systems, *JIP*, **1**, 4 (1979) 187–192.
3. Linch, H. W. and Page, J. B. The OS/VS 2 Release 2 System Resources Manager, *IBM Syst. J.*, **13**, 4 (1974) 274–291.
4. Bernstein, A. J. and Sharp, J. C. A Policy-Driven Scheduler for a Time-Sharing System, *Commun. ACM*, **14**, 2 (1971) 74–78.
5. Muntz, R. R. Analytic Modeling of Interactive Systems, *Proc. IEEE*, **63**, 6 (1975) 946–953.
6. Muntz, R. R. and Wong, J. Asymptotic properties of closed queueing network models, *Proc. 8th Annu. Princeton Conf. Inf. Sci. Syst.* (1974) 348–353.
7. Kleinrock, L. Certain analytic results for time-shared processors, *Proc. IFIPS* (1968) 838–845.
8. Baskett, F., Chandy, K. M., Muntz, R. R. and Palacious, F. G. Open, closed and mixed networks of queues with different classes of customers, *J. ACM*, **22**, 2 (1975) 248–260.
9. Gordon, W. J. and Newell, G. F. Closed queuing networks with exponential servers, *Oper. Res.*, **15** (1967) 254–265.
10. Chang, W. Preemptive priority queues, *Oper. Res.*, **13** (1965) 820–827.
11. Davis, R. H. Waiting time distribution of a multi-server, priority queuing system, *Oper. Res.*, **14** (1966) 133–136.
12. Denning, P. J. The Working-Set Model for Program Behavior, *Commun. ACM*, **11**, 5 (1968) 323–333.
13. Denning, P. J., Kahn, K. C., Leroudier, J., Potier, D. and Suri, R. Optimal Multiprogramming, *Acta Inf.*, **7** (1976) 197–216.
14. Noguchi, K. and Motooka, T. An Analysis of Traffic Control in Time Sharing Systems, EC67-15(1967-09), IECE Japan.
15. Sevcik, K. C. Priority scheduling disciplines in queueing network models of computer systems, *Proc. IFIP* **77** (1977) 565–570.
16. Hitachi, Ltd. HITAC H-8549-2 Disk Driver, H-8589 Disk Controller, Hardware Manual, 8080-2-007 (1974).
17. Bard, Y. The modeling of some scheduling strategies for an interactive computer system, Proc. Int. Symp. Comput. Perform. Model. Measur. Eval. (1977).
18. Yoshizawa, Y., Nauchi, T., Hasegawa, Y. and Inada, N. Performance Evaluation for COMTRAC-H Traffic Control System, *Trans. Inf. Process. Soc. Japan*, **20**, 2 (1979) 179–186.

(Received August 21, 1879: revised April 16, 1980)