An Experiment on the General Resources Manager in Multiprogrammed Computer Systems

TOHRU NISHIGAKI*, CHIAKI IKEDA*, KAZUHIKO OHMACHI*, and KENICHIRO NOGUCHI**

This paper presents experimental results on a new resource scheduling algorithm for balanced response/throughput optimization. The scheduling principle is based on a feedback concept, which keeps the utilization of each resource and the service of each process within prespecified ranges.

Although this principle has been introduced by the OS/VS2 Release 2 System Resources Manager (SRM), it uses only swapping as a basic control means. This makes the SRM ineffective in an undercommitted real memory environment.

The new algorithm, termed the General Resources Manager (GRM), controls performance by adjusting the allocation priority for a central processing unit and channels, as well as swapping.

Measurements on the empirical implementation have revealed high performance for the GRM in various time-sharing/batch environments, as compared with the algorithm which employs only swapping.

1. Introduction

Recently resource scheduling algorithms based on a feedback concept have begun to be in use [1][2][3]. They are considered very promising to satisfy two performance objectives—response and throughput. Such an algorithm periodically monitors the service supplied to each process, and control so as to keep it within an acceptable range. Bernstein and Sharp proposed this kind of algorithm [2][3] which can attain a satisfactory response/turn around time, by introducing a service function called a "Policy Curve". However, to our knowledge, a feedback scheduling algorithm addressing a balanced achievement of the two objectives was first implemented in the OS/VS2 Release 2 System Resources Manager [1]. The System Resources Manager (SRM) improves both throughput and response by keeping not only service of a process but each resource utilization within a prespecified limit. For this purpose, the SRM employs swapping (real memory allocation), as a basic control means. The use of swapping, however, confines the applicability of the algorithm. This is because swapping is effective only in overcommitted real memory environments. Apparently, as processes' working-set sizes change, this condition is not always met. This has inevitably necessitated an algorithm which can cope with diverse environments.

A new resource scheduling algorithm is developed. This algorithm, the General Resources Manager (GRM), controls performance by not only swapping but scheduling of various resources such as channels and a central processing unit. By paying attention to "constraints on resource allocation", the GRM associates resource allocations with each other, that had previously been done independently. During this procedure, the system state is monitored periodically, and the allocation mechanism is adjusted when a bottleneck occurs with the resource.

The purpose of this paper is to reveal the improved conditions that are possible for the GRM with respect to response and throughput in undercommitted as well as overcommitted real memory environments. An empirical implementation of the GRM made it possible to obtain some measurement results. In addition, simulations were carried out as a supplement.

2. Definition of States and Objectives

The problem is to assign resource allocation priorities in such a way as they attain balanced response/throughput optimization. A general control policy has to be given for each system state to achieve the objectives. To begin with, states and objectives are defined.

(1) The objective of a resource utilization control

A "resource" means a logical unit. A channel is a logical channel and a CPU is a unique resource regardless of the actual number of processours utilized. As for real memory, a working-set [5] is treated as the unit of allocation, instead of each page frame.

Let u_j be the utilization of resource j and ρ_j be the lower limit of the acceptable range. Resource j is defined to be either "busy" if $u_j \geq \rho_j$, or "idle" if $u_j < \rho_j$. Consequently, the system with n resources has 2^n distinct states. The control objective is given by eq. (1).

$$u_j \geq \rho_j \quad (j=1, 2, \cdots, n) \tag{1}$$

A resource service requester is called a "transaction".

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

A batch job or a terminal-initiated command becomes a transaction. In order to identify a transaction for resource utilization control, the resource use characteristic v_{ij} is defined as follows: v_{ij} is the percentage amount supplied to transaction i in the total capacity of resource j where no resource contention is assumed. Transaction i is called "a heavy user of j" if v_{ij} is of a large amount.

(2) The objective of service distribution control

Resource service count of transaction i is accumulated resource service amount that has been supplied to transaction i. It is measured in the service units (s.u.) and calculated by eq. (2).

$$R_i(\tau)=\sum_j e_j \cdot r_{ij}(\tau). \tag{2}$$

Transaction age τ represents the elapsed time measured from the i's arrival. $r_{ij}(\tau)$ represents the service amount of resource j which has been provided to i during τ. e_j is its weight. The Policy Driven Scheduler [2][3] attempts to keep $R_i(\tau)$ above a Policy Curve $f(\tau)$. However, the scheduling algorithm presented here employs a Performance Objective g introduced by the SRM [1] as a service objective function. This is because g is more adaptable to changes in the workload [1][2][3]. g specifies an ideal resource service rate as a function of the workload level. It is assumed that g is a monotonously decreasing function.

Let S_i be transaction i's resource service rate (=the resource service amount that is being supplied to transaction i per second.) and g be the Performance Objective which is associated with i. Here, S_i is given as the slope of $R_i(\tau)$. The Normalized Workload Level (NWL) of i is given by the following equation (See Fig. 1):

$$\mathrm{NWL}_i=g^{-1}(S_i), \tag{3}$$

where g^{-1} represents the inverse function of g.

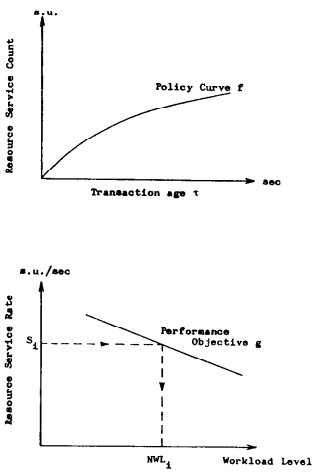


Fig. 1 Service objective function: Policy Curve, Performance Objective.

The state can be defined by each transaction's NWL. The control objective is the minimization of the NWL's deviation, as shown in eq. (4):

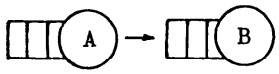
$$\sum_i (\mathrm{NWL}_i-\overline{\mathrm{NWL}})^2 \rightarrow \min, \tag{4}$$

where $\overline{\mathrm{NWL}}$ represents the mean of NWL. This objective ensures the supply of the relative rather than the actual resource service rate to each transaction.

3. The General Resources Manager

The general control policy is an integrated priority assignment scheme that is utilized to attain eq. (1) and (4). In this policy, the assignment of an allocation priority for a resource is associated with those for other resources. The concept of "constraints on resource allocation" is used to obtain the policy.

In such a case where a transaction has to be allocated resource A in order to request the service of resource B, A is called B's predecessor, and B is called A's successor.* The relation is expressed as follows:



Assuming that A is "busy" and B is "idle" the utilization of B can be improved by giving precedence to B's heavy users in the allocation of A. In addition, the deviation of NWL can also be controlled by adjustment of the allocation priorities for A. This is because a "busy" resource is a key resource in service distribution.

The constraints on the allocation for CPU, channels, real memory, and virtual memory are depicted in Fig. 2. For example, if the real memory is "busy" and the CPU is "idle", the CPU utilization can be improved by swapping-in CPU-bound transactions. Swapping is also effective for controlling response/turn around time. On the other hand, if the CPU is "busy" and the channels are "idle", the utilization of the channels can be improved by increasing the dispatching priorities of the I/O bound transactions. In this case, the adjustment of dispatching priorities is also effective for controlling the service distribution.

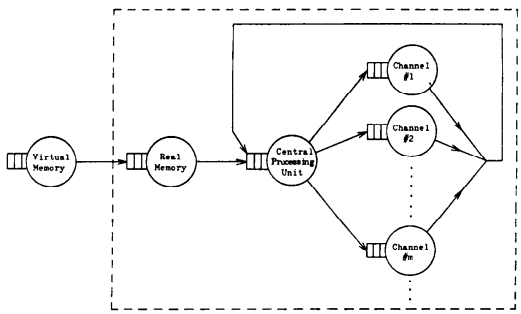


Fig. 2 An example of resource allocation constraints.

* Naturally a successor of B is also A's successor.

The control objective given by eq. (1) is re-defined by eq. (5), because only CPU and channels have to be busy for throughput improvements.

$$u_j \geq \rho_j \quad (j=\text{CPU, CH}) \tag{5}$$

The above discussion leads to the following general control policies to achieve eq. (4) and (5) in each state.

- The resource utilization control policy:

Increase the priorities of an "idle" resource's heavy users, for allocation of such resources as are "busy" and predecessors of the "idle" resource.

- The service distribution control policy:

Increase the priorities of those transactions which have relatively large NWL and decrease ones of those which have relatively small NWL, for allocation of "busy" resources.

Priorities are computed periodically, subject to the control policies mentioned above. Here, resources enclosed by the dashed line in Fig. 2 are considered.

Transaction i's priority for a real memory allocation $P_i(\text{RM})$ is given by eq. (6). However, no swapping is done if the sum of all transactions' working-set sizes does not exceed the real memory capacity; i.e., real memory is "idle". In such cases, the real memory is allocated to all transactions.

$$P_i(\text{RM})=\alpha_M \cdot \text{CP}_i(\text{RM})+\beta_M \cdot \text{IO}_i(\text{RM})+\gamma_M \cdot \text{WL}_i(\text{RM}). \tag{6}$$

α_M, β_M and γ_M in eq. (6) are weights for balancing the response and throughput. Note that CPU and channels are successors of the real memory. For simplicity we assume transaction i uses only channel#m. Each term of eq. (6) is given as follows:

$$\begin{cases} \text{CP}_i(\text{RM})=v_{i,\text{CPU}} \cdot \delta_{\text{CPU}} \cdot (\rho_{\text{CPU}}-u_{\text{CPU}})^2 \\ \text{IO}_i(\text{RM})=v_{i,\text{CHm}} \cdot \delta_{\text{CHm}} \cdot (\rho_{\text{CHm}}-u_{\text{CHm}})^2 \\ \text{WL}_i(\text{RM})=(\text{NWL}_i-\overline{\text{NWL}}) \cdot |\text{NWL}_i-\overline{\text{NWL}}| \end{cases} \tag{7}$$

$$\delta_j=\begin{cases} 1: u_j<\rho_j \\ 0: u_j \geq \rho_j \end{cases} \quad (j=\text{CPU, CHm}) \tag{8}$$

u_j, v_{ij} and NWL_i are periodically monitored. v_{ij} is computed as follows, where transaction i is assumed to use channel#m $\Delta t'$, at every Δt of CPU execution.

$$\begin{cases} v_{i,\text{CPU}}=100 \cdot \Delta t/(\Delta t+\Delta t') \\ v_{i,\text{CHm}}=100 \cdot \Delta t'/(\Delta t+\Delta t') \end{cases} \tag{9}$$

Transactions with high priority are swapped-in, and allocated the real memory of their working-sets.

The algorithm described above is based on an idea which has been introduced by the SRM. In other words, the SRM loses its controllability when the real memory is mostly idle.

Transaction i's priority for CPU allocation $P_i(\text{CPU})$ and channel#m allocation $P_i(\text{CHm})$ are given by eq. (10) and (11) respectively. However, the priority is changed only when each resource is "busy".

$$P_i(\text{CPU})=\alpha_p \cdot \text{IO}_i(\text{CPU})+\beta_p \cdot \text{WL}_i(\text{CPU})$$

$$\begin{cases} \text{IO}_i(\text{CPU})=(1-\delta_{\text{CHm}}) \cdot \text{OLDCHm} \\ \qquad\qquad +\delta_{\text{CHm}} \cdot \zeta \cdot v_{i,\text{CHm}} \cdot (\rho_{\text{CHm}}-u_{\text{CHm}}) \\ \text{WL}_i(\text{CPU})=\eta \cdot (\text{NWL}_i-\overline{\text{NWL}}) \end{cases} \tag{10}$$

$$P_i(\text{CHm})=\alpha_c \cdot \text{CP}_i(\text{CHm})+\beta_c \cdot \text{WL}_i(\text{CHm})$$

$$\begin{cases} \text{CP}_i(\text{CHm})=(1-\delta_{\text{CPU}}) \cdot \text{OLDCP} \\ \qquad\qquad +\delta_{\text{CPU}} \cdot \nu \cdot v_{i,\text{CPU}} \cdot (\rho_{\text{CPU}}-u_{\text{CPU}}) \\ \text{WL}_i(\text{CHm})=\xi \cdot (\text{NWL}_i-\overline{\text{NWL}}) \end{cases} \tag{11}$$

Note that channel#m is a successor of the CPU and the CPU is also a successor of channel#m. α_p, β_p, α_c and β_c are weights for balancing the response and throughput. ζ, η, ν and ξ are positive constants. OLDCHm in eq. (10) and OLDCP in eq. (11) represent $\text{IO}_i(\text{CPU})$ and $\text{CP}_i(\text{CHm})$ computed last time respectively. They are used because the priority modification for the resource utilization control should not be done when resources are "busy".

The CPU is dispatched to the "ready-to-execute" transaction with the highest priority P(CPU). The dispatching algorithm of eq. (10) shares its objective with the dynamic dispatching control [6]—improved use of channels. However, the dynamic dispatching control has no control over service distribution. The service distribution based on eq. (10) conceptually includes the traditional time-slicing control [4].

I/O operations requested by the transactions are executed in the order of decreasing priority P(CHm). The only difference between the CPU and channel allocation is that the CPU is preemptive while channels are not.

4. Experiments

4.1 Measurements

The GRM was implemented on our large scale computer M-180. As the first step of the implementation, the allocation priorities of the two resources, real memory and CPU, are determined by the GRM control policy. The average computing interval for P(RM) and P(CPU) are about 4 sec and 2 sec respectively. Since we chose rather CPU-bound workload in the experiment, the omission of the channel scheduling is not considered to affect the measurement results. The study on the GRM including the channel scheduling would be our further research concern.

As was mentioned above, the SRM constitutes a subset of the GRM. Therefore it is possible to compare the performance for the GRM with that for the SRM (Here the real memory scheduling in the SRM is assumed to be the same as the one in the GRM, consequently the SRM compared is not precisely the same as the OS/VS2 Release 2 SRM.). As for CPU scheduling in the SRM, it is assumed that the priorities of batch

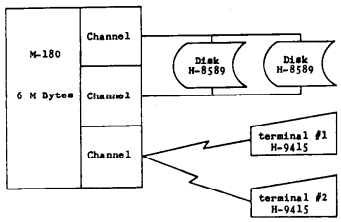


Fig. 3 The hardware configuration used in the experiment.

transactions are determined by the dynamic dispatching control [6], while those of terminal-initiated transactions are constant. The constant priority of terminal-initiated transactions is specified to be higher than any of the batch transaction's priority, in order to avoid the degradation of the terminal-initiated transaction's response.

The measurement was composed of two parts. The part I concerns the response of the terminal-initiated interactive transactions and the part II concerns the service distribution among batch transactions. Both were carried out on the M-180 system with 2 terminals (See Fig. 3). The real memory capacity can be extended up to 6 M Bytes which yields completely undercommitted real memory environment; i.e., the environment where the real memory is mostly idle.

(1) The Measurement Part I

The two terminals were always active throughout this measurement. The terminal #1 initiated short interactive transactions by "EDIT" command with think time interval of about 20 sec. On the other hand, the terminal #2 initiated a long executive transaction; i.e., the compilation & execution of FORTRAN job of which the execution phase requires 10 sec CPU execution without any I/O operations. This transaction was repeatedly initiated by the terminal #2 until the end of the measurement.

In addition to these terminal-initiated transactions, 2 batch transactions were processed concurrently. They are both the compilation & execution of FORTRAN job with program size of 250 K Bytes.

The specified service objective functions are shown in Fig. 4. Here R represents the resource service count of a transaction. The service objective functions in Fig. 4 are considered standard, where preference is given to "short" rather than "long" transactions, to terminal-initiated rather than batch transactions.

The histogram of short interactive transaction's response time is depicted in Fig. 5, which was observed during the 480 sec measurement interval.

For the GRM, the dispatching priority of a "long" transaction initiated by the terminal #2 was modified based on eq. (10), as it received CPU service. On the other hand, it was kept constant for the SRM, making this difference. To modify its priority by the dynamic dispatching control, however, never solve the lackness of service distribution control for the SRM. This is

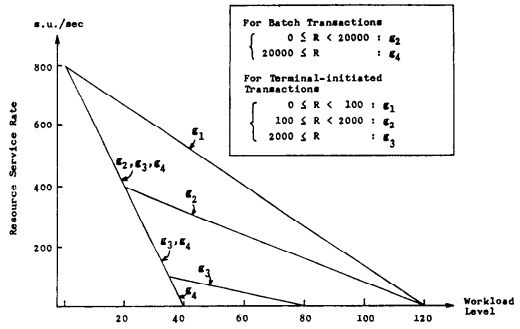


Fig. 4 The service objective functions specified in the Measurement Part I.

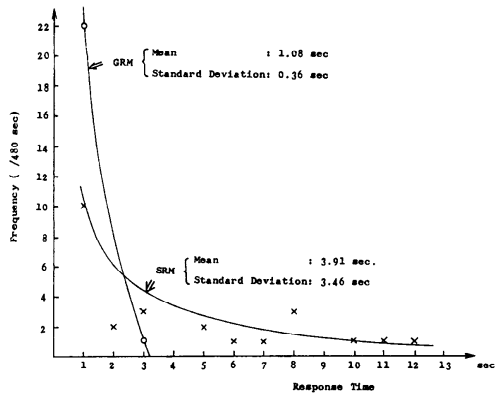


Fig. 5 Performance comparison of the two schedulers with respect to the responsiveness of interactive transactions initiated by terminal #1.

because the dynamic dispatching control aims only at the increase in throughput and might cause serious imbalance in service distribution.

(2) The Measurement Part II

In this measurement, no transactions were initiated by the terminals and only 6 uniform batch transactions were processed in parallel. Any of them is the execution of the FORTRAN job which requests I/O operations at about every 30 K steps and real memory of 25 K Bytes each. We divided these 6 transactions into two groups. The three of them (#1 ~ #3) were associated with the service objective function g_1, and the rest of them with g_3 (See Fig. 6). The former group should be supplied with more resource service than the latter, and the accuracy of the service distribution control is reflected by the standard deviation of NWL.

The change in service distribution controllability for the GRM as a function of weight β_p/α_p is depicted in Fig. 7 (Here, weight β_M/α_M was fixed at 1.0). Note that the dynamic dispatching control [6] corresponds to the case $\beta_p/\alpha_p=0$, where preference is always given to I/O bound transactions (Since all transactions were uniform in this measurement, they were given the same dispatch-

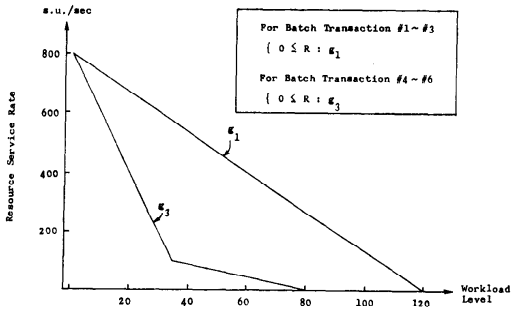


Fig. 6 The service objective functions specified in the Measurement Part II.

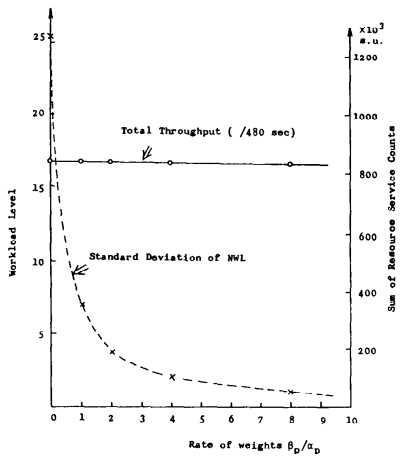


Fig. 7 Effect of the GRM performance control on service distribution/throughput.

ing priority.). The standard deviation of NWL indicates that the increase in β_p/α_p improves service distribution controllability. On the other hand, the insignificant degradation in throughput (=total resource service amount supplied to transactions) was caused by the uniformity of the transactions.

In a general way, the increase in β_p/α_p is likely to degrade throughput; especially in such a case as high service objective functions are specified for CPU bound transactions and vice versa for I/O bound transactions. Apparently no resource scheduler can always achieve two potentially conflicting performance objectives, response (service distribution) and throughput. Therefore a resource scheduler should be judged by the degree how it can control performance (response/throughput), rather than performance itself.

4.2 Simulations

There can be many active terminals in actual cases and the change in working-set size causes overcommitted as well as under-committed real memory environments. Supplementary simulations were carried out to study performance for the GRM in those environments.

The simulated hardware configuration assumed 40 active terminals and 1.6 M Bytes real memory capacity. Each of 40 terminals was assumed to initiate either Interactive (I)-type transactions or Executive (E)-type transactions at every 20 sec [8] think time interval. To make various real memory environments, we simulated 8 batch transactions together with terminal-initiated transactions. Their working-set sizes w changed from 160 K Bytes to 240 K Bytes, thereby making under-committed (when $w=160$ K Bytes) and overcommitted (when $w=240$ K Bytes) real memory environments. Half of them were CPU bound and the rest were I/O bound, and requested I/O operation at every 100 K steps and 10 K steps respectively. The E-type transactions were assumed to have the same characteristics as CPU bound batch transactions. As for the I-type transactions, the execution requirement was set at 100 K steps [7] with working-set size of 80 K Bytes [7]. The specified service objective functions in the simulations were standard ones like Fig. 4.

The responsiveness of the I-type transaction is depicted in Fig. 8, which was observed in the simulated interval of 300 sec. (Here, response time denotes internal processing time; i.e., it does not include the message's line transmission delay.). It is indicated in Fig. 8 that the responsiveness for the GRM is quite stable although that for the SRM degrades as the number of such terminals increases as initiate E-type transactions.

The difference in service distribution controllability between the GRM and the SRM is also shown in the standard deviation of NWL in Fig. 9. This figure indicates that the improvements in service distribution control is not necessarily accompanied with the degradation in throughput.

In summary, the difference in performance control

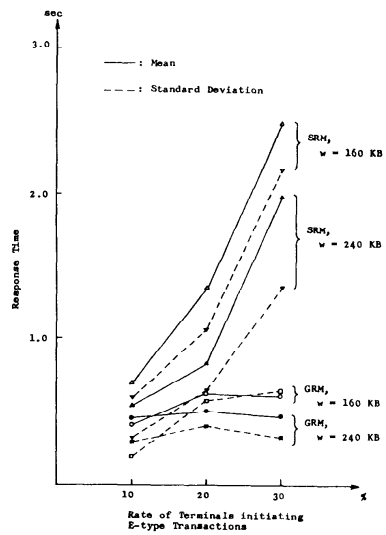


Fig. 8 Simulated effect of terminal characteristics on the I-type transactions' responsiveness.

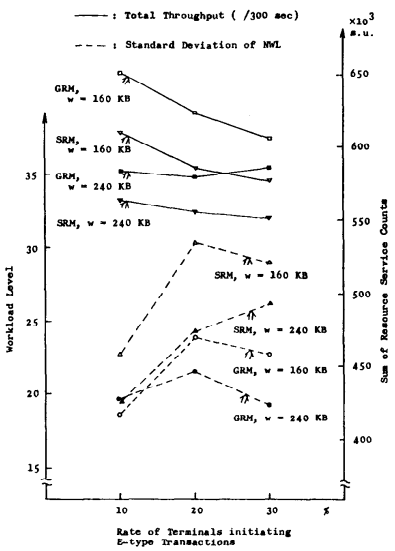


Fig. 9 Simulated effect of terminal characteristics on service distribution/throughput.

between the GRM and the SRM decreases in an over-committed real memory environments, but it is still significant.

5. Conclusion

The General Resources Manager (GRM) gives an integrated resource scheduling scheme. The GRM recognizes the bottleneck resource in performance, and adjusts its allocation priorities. This enables the GRM to improve both response and throughput in diverse environments. The GRM conceptually includes the OS/VS2 Release 2 System Resources Manager [1] which uses only swapping, the dynamic dispatching control [6], and the traditional time-slicing control.

Measurements on the empirical implementation and supplementary simulations have revealed satisfactory performance controllability for the GRM. This is especially true, when compared with the scheduling algorithm which uses only swapping.

Acknowledgements

The author would like to thank Dr. Setsuo Ohsuga of the University of Tokyo, for his helpful discussions and comments.

(Received October 12, 1978)