A SIMULATION STUDY OF ADAPTIVE SCHEDULING POLICIES IN INTERACTIVE COMPUTER SYSTEMS

Samuel T. Chanson and Craig D. Bishop

Department of Computer Science,
The University of British Columbia

INTRODUCTION

Recently, some work has been done in the area of dynamically adaptive scheduling in operating systems (i.e., policies that will adjust to varying workload[1] conditions so as to maximize performance) [4], [5], [10], [11]. However, most studies deal with batch-oriented systems only. The University of British Columbia operates an IBM 370/168 running under MTS (Michigan Terminal System) which is principally used interactively. It has been known for some time that the system is Input/Output bound. The main goal of this work is to determine to what extent adaptive control, particularly as related to processor scheduling, can improve performance in a system similar to U.B.C.'s. Simulation is used throughout the study and because of this, the simulator and the workload are described in some detail. The target machine is a somewhat simplified version of the U.B.C. System.

WORKLOAD MODEL

Since the performance of a computer system depends not only on the system but also on the workload that drives it, part of the research is in the development of a suitable workload model. The basic requirements of the model are: 1) it should be realistic (i.e. closely represent the actual workload not only in total system resource demand but also in the order of these demands), 2) it should be convenient to use (i.e., the real workload can be easily expressed in terms of the model and the model is readily modifiable), 3) it should be compact and concise (i.e., the execution of the workload does not require excessive CPU time). The real workload itself is not a suitable model because it is not concise, nor is it easily modified for experimentation. The traditional resource demand model [9], where the workload is expressed in terms of gross resource demands often drawn from distribution curves, is not acceptable because it is not realistic. In an interactive multiprogramming environment, where many processes are competing for limited resources, the order in which the demands arrive significantly influences the performance of the system and most resource demand models do not model this well. The problem is in the assumption of the independence of the statistical distributions used. What is needed are joint distributions relating the different variables, but these are not practically constructable because of the complexities of interactions amongst these variables.

Our workload is expressed in a language in between a high level language and the conventional resource demand model and is based primarily on [6]. It allows the specification of CPU and I/O device usage through assembler style commands. Each I/O command specifies a device class, type of operation, information blocks to be transferred and the associated CPU usage. Other instructions allow repetitive constructs, user terminal interaction and strict CPU usage to be specified. A given real workload can be translated into this workload specification in a straightforward manner (see [6]). The model preserves as much information as possible about the complex interrelations between various demand parameters, while allowing the workload to be expressed in a more concise form than the real workload itself.

To reduce the overhead associated with the use of a page reference string to determine page faults, page fault behaviour is simulated by a distribution which the user can specify (see below).

To construct the workload, the MTS users at U.B.C. were monitored and classified into 6 categories according to their resource demand characteristics:

(1) system programmers (fairly heavy CPU and I/O users)

(2) application programmers with heavy CPU demands

(3) beginning students (jobs are small and usually little or no file activity is involved)

(4) users whose main activity is creating and editing files

(5) the typical moderate users who will edit a file, compile it, and execute it.

(6) users characterized by large think times and small CPU and I/O demands

A number of jobs for each class of user are constructed based on their monitored request patterns.

---

(1) Workload is defined as the collection of programs, data and commands users input into the system to satisfy their information processing needs.

The aggregate of these jobs forms the standard workload for subsequent experiments.

THE SIMULATOR

The simulator [3] is written in GPSSV interfacing with the user through a high level programming language. It consists of three main modules:

(1) the system specifier, which accepts a description of the hardware configuration and characteristics of the system in English-style statements. Thus, one can specify the amount of main memory, the number of processors, the number and hardware characteristics of each type of disk and drum, the number of device controllers and channels and their interconnections, the chosen algorithms under test, etc.,

(2) the workload definition assembler, which accepts a description of the workload as outlined in the previous section,

(3) the main simulator, which simulates the execution of the workload on the hardware specified and produces various statistics about selected performance indices.

The major global performance index chosen is the distribution of response times[2] as this is the factor which is of prime importance to the interactive user. Other statistics include the utilization factors of the various system resources such as processor(s), channel(s), device controller(s), I/O device(s) and main memory as well as queue statistics of the tasks awaiting these resources. Because our system is know to be I/O bound, the entire I/O system is simulated in detail. The locations of files on disk as well as the positions of the read/write heads are maintained so that seek and latency times can be accurately determined. The sequence of requests for channels, controllers and devices are properly simulated and conflicts are resolved in the manner described in [1]. Three major simplifications are made:

(1) the privileged task[3] is not simulated

(2) the only I/O devices simulated are disks, drums, and terminals

(3) page faults are generated from a user specified distribution rather than by page reference string simulation because of the costs involved.

A consequence of the last point is that the page fault behaviour of all processes is statistically identical and is independent of other system resource demands. This can have an adverse effect on the validity of simulated results in certain areas of study (see Conclusion). Further work is being done on the efficient and accurate simulation of paging.

For details on the structure of the simulator see [3].

VALIDATION OF THE SIMULATOR

This is an extremely difficult problem. We have not been able to prove that the simulator functions correctly under all conditions, but we feel that it models actual events reasonably closely.

Each section of the simulator has been individually tested with input workloads whose characteristics produce predictable results so that simulated results can be compared with predicted performance. The results of queueing theory were compared against simulated results of some queueing systems in the model and this led to the discovery of several errors. After the initial debugging stage, a workload was constructed consisting of 45 jobs (equivalent to a medium workload on the U.B.C. system) using the method outlined earlier. This workload was then used to drive a simulated U.B.C. MTS system. The simulated results were then compared to the known system behaviour under similar workload conditions. The differences in performance characteristics were always less than 15% which is quite acceptable. The known system characteristics include various resource utilization factors, mean queue lengths for resources, and the rates of disk and drum usage; however, the response time distribution is not kept by the computing centre and is not available for comparison. We feel though that the simulated response time distribution is quite reasonable.

This process was then repeated with a workload composed of 63 jobs (representing a heavy workload) and the results obtained were equally encouraging. Then experiments were performed with different scheduling policies using the heavy workload since it is under this condition that performance is particularly important.

THE INTERACTIVE ENVIRONMENT

We observe that in an interactive environment, since a job[4] is broken into a sequence of small tasks (requests), the requests from compute and I/O bound

---

(2) Response time is defined as the interval between the completion of a command input to the computer system and the output of the first meaningful response as a consequence of that command.

(3) In MTS if the number of pages requested by a task exceeds a certain limit, and if the number of privileged tasks has not exceeded a maximum, then the task becomes privileged and can have unlimited pages in main memory. It also gets a much larger time quantum based on the number of priviledged tasks in the system at the time.

(4) a job in an interactive environment is defined in this paper as the set of interactions from a terminal from sign-on till sign-off.

jobs at any instant may be very similar in characteristics. Furthermore, these characteristics may change abruptly from request to request even for the same job. Page faults often reduce a single large CPU request into several smaller ones interleaved with paging drum I/O. It is therefore much more difficult to predict the characteristics of individual requests whereas it is relatively easy to predict the behaviour of entire jobs in batch-oriented systems. Since almost all adaptive algorithms are based on the assumption that the request pattern of the immediate future is similar to that of the recent past, the performance improvement attained by changing the software alone (e.g., using a different resource management algorithm) as opposed to that gained by hardware changes (e.g., the addition of more memory) is relatively small (up to about 15%). The exception to this is when one uses an extremely inefficient algorithm such as round robin with an infinite time quantum as the basis for comparison. Table 5 exemplifies this. In an interactive system, minimizing the mean response time should not be the only concern. As the majority of requests are small and users with large requests are more willing to wait, there should be some guarantee that short requests will be satisfied within a reasonably short time (0.5 sec., for example). Hence an important index should be the mean of those response times less than the X percentile for some X. The average of the response times less than the fiftieth percentile [denoted by RES(50%)] is chosen as another global performance index in this paper. A good algorithm should not reduce the mean response time at the expense of RES(50%).

ADAPTIVE SCHEDULING POLICIES

MTS uses basically a round robin (RR) scheduling algorithm [2] with a fixed quantum (set to 10 msr at U.B.C.). Tasks blocked for I/O (including page faults) will rejoin the head of the ready queue so as to favour I/O bound interactive jobs and ensure that tasks requiring less than one time quantum will pass through the queue only once.

A. ROUND ROBIN WITH VARIABLE QUANTUM LENGTH (RRVQ)

It seems logical that when the number of tasks in the RR queue is small, a larger quantum should be assigned. This would reduce the number of process switches and improve throughput, yet maintain the response time of small tasks at an acceptable level. On the other hand, if the number of ready tasks is large, the quantum should be reduced to keep the response time of short tasks low. Of course upper and lower limits on the quantum must be established. Some work has been done in this area for batch-oriented systems [5] and a considerable improvement in throughput under certain conditions is reported.

The most obvious implementation of this concept in an RR scheduling algorithm is to recompute the quantum after every interval T according to the expression:

$$\max[Qmin, Qmax/L] \quad (A)$$

where L = number of tasks in the ready queue
Qmin = a constant = time quantum lower bound
Qmax = a constant = time quantum upper bound

This method is unable to adjust to the instantaneous workload change. For example, if at time T1 there are only 3 tasks in the ready queue, the time quantum will be set to Qmax/3. If 20 short tasks then join the ready queue immediately, each will still receive a quantum of Qmax/3 instead of the more appropriate Qmax/23 or Qmin (whichever is larger) until T runs out. Even so, all tasks executed before T runs out will get an inappropriately large quantum even though the queue length has increased at the time the tasks are scheduled. The scheme described below will overcome this deficiency and is easier to implement because in the previous scheme the expiration of T need not coincide with the termination of a task.

The basic RR scheduling algorithm is modified so that when a task is to be removed from the ready queue and assigned the processor, it is given a time quantum according to expression (A) where L, Qmin and Qmax are as before. Thus the time quantum is computed for each task based on the instantaneous workload at the time the task is assigned to the processor. It can be shown that this scheme performs at least as well as the previous one. Despite this, however, the algorithm is unable to deal with unusual circumstances such as when a large number of short requests joins the ready queue right after the single task in the queue is assigned Qmax. In this case, the short tasks are delayed by Qmax instead of Qmin as they would have been if they had arrived slightly earlier. Another example is when there are only relatively large jobs in the system and a large quantum should be used for all of them [5]. This scheme, however, is unable to predict the size of jobs. These are inherent weaknesses in algorithms that do not discriminate between different job types. To overcome this a radically different approach must be used. The performance improvement over the basic MTS RR scheduling algorithm is small (Table I). Notice that if the values of Qmin and Qmax are not set properly, the mean response time will actually go up. However, this sequence of simulation results does show that varying the quantum length with respect to the number of tasks awaiting the processor can reduce the global mean response time as well as the response time for small tasks.

TABLE I

Performance of RRVQ with different values of Qmin and Qmax.

POLICY	QMIN (MS)	QMAX (MS)	RES (SEC)	RES(50%) (SEC)	#PROCESS SWITCHES
RR	-	-	1.81	.31	227/SEC
RRVQ	10	60	1.83	.36	196/SEC
RRVQ	10	50	1.77	.32	209/SEC
RRVQ	10	40	1.75	.30	211/SEC
RRVQ	10	30	1.85	.33	219/SEC
RRVQ	8	40	1.78	.30	229/SEC

RES = mean global response time
RES(50%) = mean of response times less than the median

B. MULTIPLE FEEDBACK DISCIPLINE (MF)

The multipline feedback discipline is also known as the shortest-attained-CPU-time first algorithm. In this algorithm, there is an ordered set of queues used for CPU scheduling. When the CPU becomes free, the task at the head of the lowest numbered non-empty queue is scheduled. If it does not complete in one time quantum, it is placed at the end of the next higher numbered queue unless the present queue is the highest numbered one, in which case the task is returned to the end of the same queue. After an I/O operation, a task is placed at the head of the queue it was last in, unless the I/O operation is a terminal interaction, in which case the beginning of a new request is signified and the task is placed at the end of the first queue.

This scheme also favours interactive and I/O bound tasks. In addition it has the effect of dynamically classifying tasks according to the amount of CPU time they have received and assigns lower priority to those that have received a large amount of CPU time. This is based on the observation that tasks receiving a large amount of CPU time have higher probability of requesting even more CPU time. In order that tasks in the upper queues will not take too long to complete, a larger quantum is usually assigned to them. A common method is to assign a quantum of length Q(1)*N**I to tasks in queue I where Q(1) is the quantum associated with the lowest numbered queue and is known as the basic time quantum, and N is a constant greater than or equal to 1 known as the multiplicative factor. Thus when a task in an upper queue gets to run (i.e., when there are no tasks in the lower queues), it is given a larger quantum. A multiplicative factor of 2 is common in MF type algorithms. We have found that the value of this factor will affect performance and that the optimal value differs from workload to workload. The optimum differs also if we refer to RES or to RES(50%).

In general, increasing this factor will increase the response times of short tasks because they may be delayed by tasks just given a large quantum. The response time of long tasks will drop so the variance of the response time of the job mix will decrease. When N is small, more medium-size tasks migrate to the top queue and are thus classified with the large tasks. Their response time will go up whereas that of long and short tasks will decrease. A detailed study to relate the optimal multiplicative factor to the workload is underway.

We believe that the multiple feedback discipline is basically a better scheduling policy than RR, and in no case is performance worse than the MTS scheduling policy (Table 2).

C. 90% RULE (NPR)

In an interactive environment, a good rule of thumb is that at least 90% of all requests should be completed within a single time quantum [7]. This ensures that the number of process switches will not be unreasonably large. In regulating so that 90%

TABLE 2

Performance of the 5-Queue MF with different Multiplicative Factor.

POLICY	N	RES (SEC)	RES(50%) (SEC)	#PROCESS SWITCHES
RR	-	1.81	.31	227/SEC
MF	.8	1.76	.28	378/SEC
MF	1.0	1.72	.27	231/SEC
MF	1.5	1.75	.28	149/SEC
MF	2.0	1.76	.29	136/SEC
MF	2.5	1.71	.29	130/SEC
MF	3.0	1.73	.30	127/SEC

N = Multiplicative Factor

RES and RES(50%) have same meaning as in Table I

of the tasks do not recycle into the second queue, queue waits will be decreased for small requests. This also ensures that large non-trivial tasks will get a reasonably-sized quantum and hence a tolerable response time. A new adaptive scheduling algorithm has been derived for use in conjunction with the MF discipline based on the above observation. The basic time quantum Q(1) is increased by a fixed amount delta Q if in the previous T units of time the proportion of tasks to leave the first queue without joining the second queue is less than L, a fixed lower bound. This proportion includes those leaving queue 1 for I/O as well as those that have finished their first quantum. If that proportion exceeds U, a fixed upper bound, then Q(1) is decreased by delta Q. If no task leaves the first queue or if the proportion lies between L and U then Q(1) is unchanged. Q(1) is not allowed to fall below Qmin or rise above Qmax. A sequence of simulations was performed with different values of L and U and the results are shown in Table 3. Notice that both global mean response time and the mean response time of small tasks decrease.

The adaptive scheduling algorithm has several desirable characteristics. Firstly, since the proportion of tasks in the higher numbered queues is more or less constant, the value of the multiplicative factor N has much less effect on mean response time. That is, it is harder to choose a bad value of N. The 90% rule is more robust (i.e., less likely to break down) than the RRVQ because it is more conservative for one thing. At each decision point, a small but fixed quantity is added or subtracted from the quantum size. Therefore a single wrong decision cannot have a great impact on system performance. Its effect can be corrected at the next decision point. More consideration is paid to the global workload condition because the scheduling decision is based on the activities within an interval of time and on global CPU demand rather

TABLE 3

Performance of 90% Rule(NPR) with different values for the upper(U) and lower(L) bounds.

POLICY	L (%)	U (%)	MPQ (%)	RES (SEC)	RES(50%) (SEC)	#PROCESS SWITCHES
RR	-	-	-	1.81	.31	227/SEC
NPR	87	93	90	1.70	.27	130/SEC
NPR	88	96	91.3	1.67	.28	128/SEC
NPR	92	95	91.4	1.68	.28	129/SEC
NPR	95	95	91.5	1.76	.29	129/SEC
NPR	92	98	94.1	1.65	.29	128/SEC

No. of Queues = 5
Mult. Factor = 2
MPQ = percentage of tasks that completed in 1 quantum
delta Q = 2ms
Qmin = 4 ms
Qmax = 25 ms
RES,RES(50%) have same meaning as in Table 1.

than on the number of tasks in the ready queue at any instant. Thus it is able to handle the situation when a large number of small tasks follows a large task. The single large task will not be given a large time quantum as it would be under RRVQ and it will quickly leave the first queue, out of competition with the small tasks. The situation of large jobs only in the system can be handled as the quantum will increase in this case. Workloads for the two situations discussed above were constructed and run under the NPR and the RRVQ. Simulation results showed that the mean response time under such situations could improve drastically (up to several hundred percents in some cases) with the NPR. The initial value of Q(1) is not important and need not be set accurately because it will be dynamically adjusted to suit the workload. In RRVQ however, since Qmax is a constant, it is important that it be set correctly. The choice of T though, requires some thought: too large a value will render the scheme insensitive to workload variation, too small a value will make the scheme unstable and increase the number of process switches. We have chosen T to be 0.5 second. An exponentially-weighted average of several time periods in the past may make the scheme even more robust. If L and U are chosen too high, the scheme becomes insensitive to variation in request pattern; if chosen too low, short tasks will suffer. For most workloads, the CPU demand can be approximated by an exponential distribution [12]. In this case, L should be about 90%, hence the name for this scheduling policy.

DYNAMIC CPU VS. I/O BALANCING (DCIOB)

It is observed that whenever there is a congestion in some queues in the system, there is a corresponding slack in other queues. The explanation of this phenomenon is simply that the number of processes competing for resources in a short period of time is more or less constant. Thus we observe that when an algorithm reduces the mean CPU queueing time, the mean I/O queueing time is usually increased, and vice versa. More work will be accomplished if the utilization of the CPU and I/O systems are balanced [8]. Thus if requests for I/O begin to build up, I/O bound tasks should not be scheduled for the CPU. Instead, tasks with large CPU demand should be scheduled reducing the probability of sending more requests for I/O to the already congested I/O system (i.e., if we cannot help the response time of I/O bound tasks, maybe we can improve it for CPU bound ones).

To test out this idea, we have to answer two questions: (1) when is the I/O system congested? and (2) which are the tasks with large CPU demand? We assume that the I/O system is congested if the number of I/O requests at any instant exceeds a certain fixed upper bound. We also assume that tasks at the higher level queues in a multiple feedback scheduling policy have higher probability of being CPU bound. Thus the multiple feedback discipline is modified to schedule the first task in the highest numbered non-empty queue if the number of I/O requests exceeds an upper bound (U). The standard MF scheduling policy is used if this number falls below a lower bound (L). The results are shown in Table 4.

TABLE 4

Performance of Dynamic CPU vs. I/O Balancing with different values of upper(U) and lower (L) bounds on number of tasks waiting for I/O.

POLICY	U	L.	RES (SEC)	RES(50%) (SEC)	#PROCESS SWITCHES
RR	-	-	1.81	.31	227/SEC
DCIOB	15	13	1.76	.28	139/SEC
DCIOB	16	15	1.75	.27	141/SEC
DCIOB	17	16	1.70	.28	140/SEC
DCIOB	18	17	1.72	.29	138/SEC
DCIOB	19	18	1.77	.29	137/SEC

(5 level MF with multiplicative factor=2)

RES,RES(50%) have the same meaning as in Table 1.

It is found that the optimal upper and lower bounds are very dependent on the characteristics of the workload and are perhaps not a good indication of I/O congestion. The U.B.C. system has 12 user

accessible disks attached to 2 block-multiplexor channels; hence the I/O wait time depends less on the total number of I/O requests than on the number of requests for the same disk (MTS uses a FCFS disk scheduling policy). The I/O wait time has dropped using DCIOB as expected, but CPU wait time is increased as the large tasks, when scheduled, may hold up the execution of short tasks. This scheme will probably work better for systems with a small number of disk drives. Nevertheless, the simulation has shown that this attempt to balance system utilization can reduce the mean response time. These results are at least as good as those in Table 2.

CONCLUSION

Several adaptive processor-scheduling policies have been presented. The multiple feedback discipline is shown to be inherently superior to the round-robin algorithm and with the addition of quantum variation using the 90% rule shows a 12% improvement over the existing MTS scheduling algorithm which is felt to be reasonably efficient already. This scheme is shown to be more robust than RRVQ and the overhead induced is comparable to that of existing non-trivial schemes. In some cases the overheads are more than compensated for by the reduced overheads of process switching. The data presented in this paper do not include overhead times. Overall CPU overhead induced by the different algorithms was estimated from the number of instructions necessary to collect and analyse the data to simulate the algorithms and from the actual CPU time spent in doing so. They are found to differ only slightly (the maximum difference was under 2% of the mean response time) for all schemes described in this paper. Furthermore, since for most schemes, data are collected and analysed each time a process switch occurs, the number of process switches per unit time is related to the amount of overhead induced. Since a process switch in a time-shared environment is always associated with paging activities, it is our opinion that a scheme which minimizes process switches is likely to induce low CPU and I/O overheads.

Incorporating information about I/O utilization into the processor scheduling policy appears to be a sound concept except that a concise and workload invariant formula for computing the expected I/O wait should be used. Table 5 compares the performance of the various algorithms discussed in the previous sections. The data represent the best simulation results of each scheme and some data have not been listed in previous tables. Since the rate of page faults is kept unchanged for different scheduling policies because the same distribution curve is used, it is felt that the actual improvement of the adaptive policies would be higher than reported here. This is because the page fault rate should decrease when the rate of process switches drops, thereby reducing response times. This study shows that adaptvie scheduling algorithms need not involve excessive overhead and can improve performance in interactive as well as batch-oriented systems.

TABLE 5

Comparison of different scheduling algorithms.

POLICY	RES (SEC)	RES(50%) (SEC)	#PROCESS SWITCHES
RR,Q=INFINITY	3.80	.60	101/SEC
MTS RR, Q=10MS	1.81	.31	227/SEC
RRVQ, QMIN=10MS QMAX=40MS	1.75	.30	211/SEC
MF, MULT.FACTOR=2.5	1.71	.29	130/SEC
DCIOB, U=17 L=16 MULT.FACTOR=2.5	1.65	.28	140/SEC
NPR,MULT.FACTOR=2.5	1.59	.29	125/SEC

RES,RES(50%) have the same meaning as in TABLE 1.

No. of Queues=5 for MF, DCIOB, and NPR.

ACKNOWLEDGMENT

The authors would like to thank John Hogg of the UBC Computing Centre systems group for information about the MTS system and D. Ferrari and the referees for their valuable comments. This work is supported by the Canadian National Research Council under contract A3554.

REFERENCE

1. IBM System/370 Principles of Operation, GA22-70000-4 File No. S/370-01, September 1974.
2. Alexander, M. T., "Organization and Features of the Michigan Terminal System," AFIPS Conference Proceedings, Vol. 40, SJCC, 1972.
3. Bishop, C., SIMCOM User's Manual, Technical Manual 76-14, October 1976. Department of Computer Science, The University of British Columbia.
4. Blevins, P. R. and Ramamoorthy, C. V., "Aspects of a Dynamically Adaptive Operating System," IEEE Transactions on Computers, Vol. C-25, No. 7, July 1976.
5. Bunt, R. B., Self-Regulating Schedulers for Operating Systems, Technical Report No. 76, January 1975, Department of Computer Science, University of Toronto.
6. Chanson, S. T., Workload Characterization of Multiprogrammed Computer Systems, Ph.D. Thesis, University of California, Berkeley, 1974.
7. Chanson, S. T. and Ferrari, D., "A Deterministic Analytic Model of a Multiprogrammed Interactive System," AFIPS Conference Proceedings, Vol. 43, NCC, 1975.

8. Denning, P. J., "Equipment Configuration in Balanced Computer Systems," IEEE Transactions on Computers, Vol. C-18, No. 11, 1969.

9. Kernighan, B. W. and Hamilton, P. A., "Synthetically generated performance Test Loads for Operating Systems," Proceedings, First SIGME Symposium on Measurement and Evaluation, February 1973.

10. Marshall, B. S., "Dynamic Calculation of Dispatching Priorities under OS360 MTV," Datamation, August 1969.

11. Ryder, K. D., "A Heuristic Approach to Task Dispatching," IBM System Journal, Vol. 9, No. 3, 1970.

12. Scherr, A. L., An Analysis of Time-shared Computer Systems, Ph.D. Thesis, Department of Electrical Engineering, M.I.T., Cambridge, Massachusettes, June 1965.