PROCESSOR SCHEDULING SIMULATION

Daniel Martin
Memphis State University, Memphis, Tennessee

A trace-driven simulator as defined by Cheng [1] is used to analyze customer service provided by a multiprogrammed computer system. In particular, several internal processor scheduling algorithms are simulated and attention is devoted to the effect these algorithms have on weighted turnaround provided to the jobs in the system.

THE SIMULATOR

The simulator is written in ALGOL-60 and is implemented on a XEROX Sigma 9 computer. The basic assumption in a trace-driven model is that jobs will enter the system and repeatedly request a burst of CPU time followed by burst of IO time. These bursts are referred to as the CPU and IO cycles respectively. Although it is possible for a job to have its own CPU and IO cycles overlapped in an actual computer system, this is not allowed in our model. CPU and IO cycles of distinct jobs are, of course, overlapped. The CPU and IO cycles can be constants, read off a file (which may have been created using a monitor on an actual system) or generated by random number generators. The results in the sequel were obtained using random number generators. The simulator is cognizant of channel conflict and supervisory overhead; it provides up to 5 levels of multilevel queuing for CPU service.

The simulator was orginally designed for use in a graduate operating systems course, where it was necessary that students be able to easily modify and create CPU scheduling disciplines. Accordingly, the queuing mechanism is located in a single ALGOL procedure. (It is most difficult to predict the effect of scattered modifications to a large program). The students worked with a simple, preliminary version of the model with great enthusiasm, 8 students managed to amass $2,399.98 of computer time in only 5 weeks.

The data required to drive the simulator consists of a set of job profiles which consist of total CPU time required, job identification number and, optionally, an entry level in a multilevel queque. (This last item can be thought of as an external priority). Further, the number of words of main memory that the computer being simulated has available for user tasks, and n, $1 \leq n \leq 10$, the number of IO channels for the simulated computer must also be provided. All IO channels are viewed as floating channels, i.e. any channel can be connected to any IO device.

The simulator maintains several lists, the dominant one being the ACTIVE list. The ACTIVE list contains information about each non-terminated job; in particular it contains the identification number, total CPU time required, CPU time received, IO time received, main memory requirement, CPU time remaining before the next IO cycle, and a timeslice, q, (as determined by the QUEUEIN procedure).

The BLOCKED list contains pointers to the ACTIVE list for all jobs performing IO. It is a linked list, sorted in order of IO complete time. The READY list correspondingly holds pointers to ACTIVE for all jobs ready to use the CPU. It is a five level list which is ordered according to whichever queuing discipline is being simulated.

Job profiles are placed in the ACTIVE list by GETPROG which then calls QUEUEIN. QUEUEIN places the jobs, or more precisely, pointers to the jobs, in the READY list. Jobs are added to ACTIVE until a job too large to fit in the remaining main memory is encountered. At this point, DISPATCH is called to detach a job from the READY list and assign it to the CPU for an amount of time which was determined by the QUEUEIN procedure. Jobs are run to completion; as each job exits, a test is made to see if the next job will fit in main memory. If so, it, and any additional jobs that will fit, are added to the READY queue. Needless to say, the memory management modeled by this simulator is extremely primitive.

QUEUEIN is called wherever a job must be inserted into the READY list, i.e. at time slice interrupts, preemptions, IO complete interrupts and new job entry. It calculates a timeslice (or a maximum amount of time the job can hold the CPU) and inserts the job in the READY list.

In order for a job to perform IO, it must possess an IO channel. Channels, if available, are assigned upon an IO initiate requests. If no channel is available, the CPU will remain idle until a channel becomes free; the requesting job will then be assigned to that channel and simulation will resume.

It is possible to forcibly detach a job from the CPU prior to a time slice or CPU complete interrupt. This process is called preemption and will only take place if (i) the global variable PREMP is set to TRUE; and (ii) a job J enters the READY queue at level j and $j <$ level of the job currently using the CPU. In this model, there is no preemption unless a multilevel queuing discipline is being modeled.

Another way a job can be detached from the CPU prior to the completion of its CPU cycle is through a time slice interrupt. The procedure QUEUEIN assigns a time slice, q, to each job; if the time remaining in the current CPU cycle is greater than the timeslice, the job will only receive the CPU for q ms. At the end of q ms of CPU service, QUEUEIN will be called and a new (possibly the same) job will be assigned to the CPU by DISPATCH.

Overhead is assigned to the CPU at each time slice, preemption, dispatch, IO complete and IO initiate request. Different constants are maintained for swaptime, IO complete and IO initiate calls. If the CPU is engaged when an overhead call occurs, the user CPU cycle is extended and time is updated to reflect the time required to handle the call. This update may result in the occurence of more supervisor calls - all of which are handled in the same manner. Of course, if the CPU is idle at the time of a supervisor call, the process is simplified. As one might suspect, this situation occurs rarely.

The data used in the simulator runs which are described below was obtained from random number generators. It has been observed [2,3] that both CPU cycle and total CPU time are highly skewed. Accordingly, lognormal random numbers were generated to use as these variables. Many simulation experiments, however, use exponential numbers as CPU cycles [4,5]. We have also generated CPU cycles which fit an exponential distribution. The results obtained with different distributions are compared in the next section. Procedures described in [6] were used to generate these numbers. Normal variables were generated to use as memory requirements.

RESULTS

Preliminary runs of the simulator used constant CPU and IO cycles within individual jobs. Gwynn [7] has shown that if this is the case and (i) these constants are generated using the mean, varience and distribution of previous runs in which non-constant cycles were used; (ii) the sched-discipline is Round-Robin, then the global results, e.g. CPU utilization and throughput are not markedly perturbed. A further simplification which provides that the same constant IO and CPU cycles be used for every job has been shown to be inadequate [5]. The simplification suggested by Gwynn does affect the local results, such as order of exit and time of exit for individual jobs. It is important for our purposes that these results remain unperturbed.

As a measure of service provided to individual customers we use weighted turnaround which has been defined by Madnick and Donovan [8] as elapsed time to complete a job divided by runalone time. Thus, a weighted turnaround of 2.0 means a job took twice as long to run as if it were running alone. Average weighted turnaround for a job stream is $\sum_{\text{all jobs}}$ (weighted turnaround)/ number of jobs.

Although external job scheduling can have a dramatic effect on the utilization of a computer system, we do not consider this aspect of scheduling in this paper. As has been done in many simulation experiments [2,5,9] we look at the effect of various internal CPU scheduling disciplines on throughput and CPU utilization. Additionally, we monitor the affect of the various strategies on service provided to individual customers, using weighted turnaround to measure customer service.

It is well-known that the scheduling of the CPU in a multiprogrammed computer system is complicated by the fact that several reasonable goals are contradictory. [3,8]. Increasing CPU utilization may adversely affect response, maximizing throughput may result in poor use of peripheral equipment. We want to look at the relationship between throughput and average weighted turnaround; we are aware that there is no a priori reason to believe that there should be any such relationship. Note that the best turnaround can always be obtained by running the jobs seperately; this, of course, is hardly attacking the root of the scheduling problem in a multiprogramming computer system.

Sherman, Baskett and Browne [2] used results from a trace-driven model to conclude that a good scheduling strategy must be preemptive and must not allow any job to seize the CPU for extended periods of time. Once these conditions were met, there was not a large change in CPU utilization or throughput. In fact, the maximum percentage increase in throughput noted after these restrictions were met was 6.97% between a random guess and a theoretical "best" policy. (The "best" policy assigns the CPU to the job which requires the least amount of CPU time at the instant of assignment; it is more than difficult to implement such a strategy in an actual computer). They also noted that even a simple Round-Robin scheduling mechanism provides throughput only marginally lower than the "best" strategy. It does not appear to be extraordinarily difficult to drive up CPU utilization and, consequently, throughput provided sufficient jobs and resources to support them are available.

In addition to classical First-Come, First-Serve (FCFS) and Round-Robin (RR) with various time slices, several multilevel scheduling disciplines were simulated. In a multilevel discipline, a job at level i will be assigned to the CPU only if (i) every job at level $j < i$ is doing IO; or (ii) there are no jobs at level $j < i$. If the job using the CPU is at level i, and it is possible for an entering the READY queue at a job at $j < i$ to seize the CPU from this job, then the strategy is preemptive. All scheduling within a single level is RR, the quantum for level i is $(i50) + 50$ ms.

In the AD (autodrop) discipline, all jobs enter the CPU at level 0 and will receive a maximum of 500 MS CPU time at that level. If they require more than 500 MS, they are demoted to level 1 where they remain until an additional 500 MS of CPU time has been received, at which time they enter level 2. Jobs can remain at level 2 for 1000 MS CPU time, level 3 for 2000 MS CPU time, and level 4 forever. The ADP discipline is the same as the AD but with preemption.

The CD (cycle drop) is an attempt to exploit the skewed distribution exhibited by the CPU cycles. As each new CPU cycle is started, the job moves to level 0. A job will remain at level i until it generates 3 time slice interrupts; it then enters level i + 1. Once entering level 4 it remains at that level until it completes the CPU cycle. The CDP is the same scheduling discipline except that preemption is allowed.

The MEM discipline favors jobs with large memory requirements. If a job requires M words of main memory it is placed in the READY queue at level $i = \mathrm{MIN}(0,[4-M/]0000])$, where [ ] is used to denote the greatest integer function. MEMP is the same scheduling discipline but allows preemption.

We have experimented with different scheduling strategies and various memory and channel arrangements. Results for a 96K, 4 channel system are representative and are depicted in figures 1 and 2.

Note that switching from exponential CPU cycles to lognormal CPU cycles has almost no affect on the performance of the various scheduling disciplines if average weighted turnaround is used as the measure. The CDP and CD disciplines are designed to exploit high varience in CPU cycles; since there is less varience in the exponential data set, these strategies do provide significantly poorer throughput with exponential data then they do with lognormal data. With either data set, MEMP simultaneously provides the best throughput, best CPU utilization and worst average weighted turnaround.

As Sherman, Baskett and Browne have predicted, the preemptive scheduling disciplines uniformly provide better throughput than non-preemptive disciplines, provided lognormal variables are used for CPU cycle time. Further, with either set of data, RR having an appropriate time slice provides good throughput; too-short quantum can, however, drive overhead up causing overall degradation of service. It is not surprising that RR does well, it is known that it will provide good results if the individual service times exhibit a large varience [2].

There does not appear to be any correlation between CPU utilization (or throughput) and average weighted turnaround. It is apparent, however, that one should be careful of modifying the CPU scheduling algorithm based only on simulation results which predict increased throughput. In the lognormal data for example, a 15.04% increase in throughput occurs when MEMP is substituted for FCFS; at the same time, however, a 43.02% increase in average weighted turnaround and 606.87% increase in the standard deviation of average weighted turnaround is observed. This large varience in the standard deviation is most galling to the user of a computer system. Needham [10] points out the danger of ignoring feedback in a situation like this; large jobs are encouraged and will be provided. The initial throughput gain will eventually be destroyed and requests for more memory in order to provide better service will soon be heard. Donovan [11] also has observed that users rapidly modify their behavior to circumvent and take advantage of policy decisions.

Several studies have placed emphasis on turnaround received as a function of total service time required [3,4]. It is commonly desired that the scheduling mechanism of a multiprogrammed computer system should provide priority service for short jobs. RR, AD and ADP disciplines have been predicted to exhibit the desired property. (Since a short job might consist of a few relatively long cycles, there is no reason to believe that CD or CDP strategies will provide such a service.) We wished to verify these results with the simulator. Alas, the RR discipline failed to provide expected priority service with either CPU distribution, in fact FCFS uniformly provided better average weighted turnarounds for short jobs regardless of the quantum given the RR discipline. The problem appears to be that jobs consist of several, rather than one, cycle and constantly feedback into the queue.

Average weighted turnaround is plotted against total service time required for several scheduling disciplines in figures 3 and 4. Regardless of CPU cycle distribution, AD or ADP strategies provide the best overall service. Additionally, these strategies provide near-best throughput and CPU utilization with both data streams.

CONCLUSIONS

We verify for the lognormal data that a preemptive, timesliced scheduling discipline provides a quantum leap in performance when compared with their non-preemptive or non-timesliced counterpart. The quantum jump is less apparent (and sometimes non-existent) with the expontential data. With no tested timeslice did any RR discipline provide better average turnaround for very short jobs than did the FCFS scheduling strategy. A multilevel feedback queue which drops jobs to lower levels (and longer time slices) as the CPU time required by the job increases provided the best mean weighted turnaround, best standard deviation of mean weighted turnaround and near-best throughput and CPU utilization.

There can be a wide differences between average weighted turnaround provided by scheduling disciplines with only minor differences in throughput. We feel this discrepancy is large enough to result in a modification of customer behavior, a resulting change in job stream characteristics and consequent non-realization of predicted performance improvement.

Our conclusions must be qualified by the fact that there is less than a consensus as to what distribution CPU cycles actually do follow.

Lognormal CPU cycles
Strategies listed in order of increasing throughput

Strategy	quantum	percent overhead	percent CPU util.	throughput jobs/hour	mean wgtd turn.	stnd. of mean wgtd turn.
FCFS	∞	2.23	74.42	292.76	1.72	.43
RR	25	5.26	77.62	301.56	1.73	.52
RR	500	2.39	77.40	303.60	1.74	.48
RR	250	2.56	78.99	309.79	1.66	.37
MEM	*	2.76	80.28	313.57	2.26	2.07
AD	*	2.65	79.37	313.60	1.37	.29
RR	150	2.79	80.13	315.25	1.64	.34
CD	*	3.24	80.42	315.40	1.68	.80
ADP	*	2.71	80.47	316.25	1.31	.34
CDP	*	3.42	81.48	316.80	1.57	.57
MEMP	*	2.89	85.72	336.88	2.46	3.08

* quantum at level i = (50*i) + 50 MS

Figure 1

Exponential CPU Cycles
in order of increasing throughput

Strategy	quantum	percent overhead	percent CPU Utd.	throughput jobs/hours	mean wgtd. turnaround	Std. dev. of mean wgtd turn.
RR	25	5.11	75.61	291.01	1.81	.43
CD	*	3.38	78.25	300.66	1.80	.73
CDP	*	3.41	77.78	301.05	1.90	1.19
FCFS	∞	2.33	78.41	301.85	1.69	.39
RR	500	2.38	78.51	302.17	1.74	.43
RR	250	2.51	78.66	302.85	1.75	.41
RR	150	2.70	78.57	303.46	1.75	.40
ADP	*	2.65	79.65	306.80	1.33	.35
AD	*	2.64	80.24	308.52	1.42	.33
MEM	*	2.94	85.93	332.12	2.36	2.63
MEMP	*	3.11	88.01	339.23	2.53	3.16

*quantum at level i = (50*i) + 50 MS

Figure 2

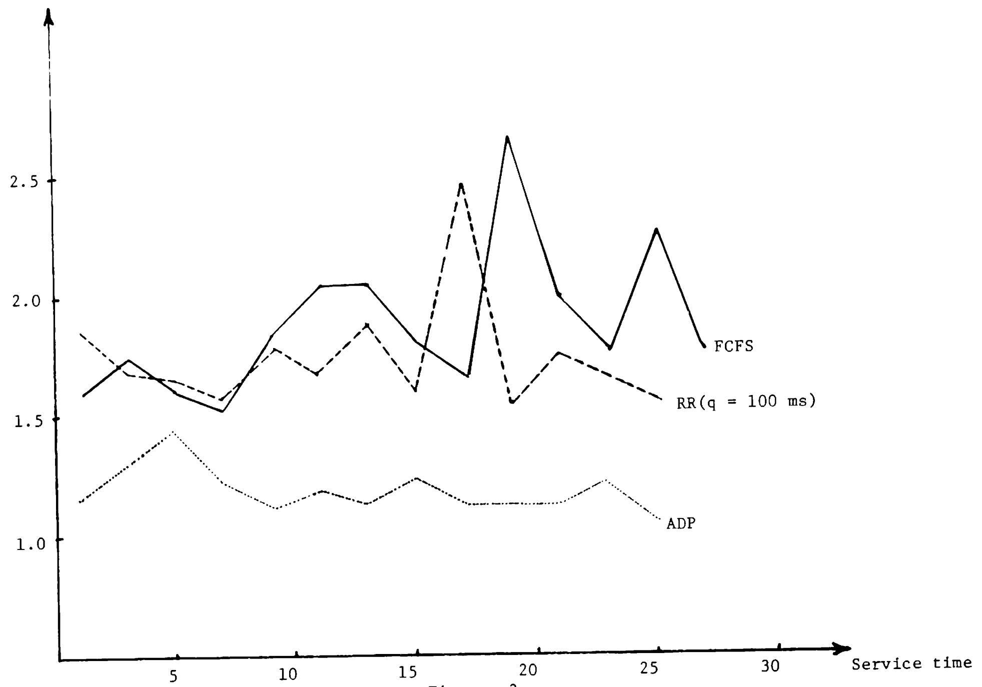

Figure 3
Lognormal Data
Service time plotted vs average weighted turnaround

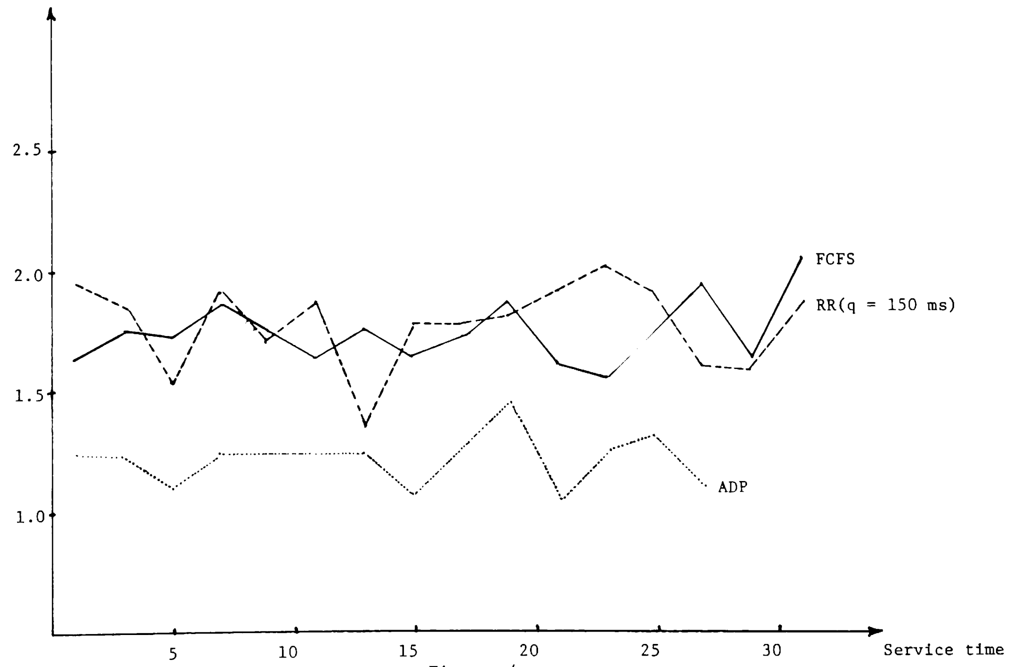

Figure 4
Exponential Data
Service time plotted vs average weighted turnaround

References

1. Cheng, P.S. "Trace-driven system modeling", IBM Sys. J. 8, 280-289, 1969.

2. Sherman, S.; Baskett, F.; and Browne, J.C. "Trace-driven modeling and analysis of CPU scheduling in a multi-programming system", Communications of the ACM 15, 1063-69, 1972.

3. Freeman, P., Software Systems Principles: A Survey, SRA, Chicago, 1975, 274-304.

4. Fuller, S. in Introduction to Computer Architecture, ed. by H. Stone, SRA, Chicago, 1975, 507-540.

5. Boyse, J. and Warn, D. "A straight forward model for computer performance prediction", ACM Computing Surveys, Vol. 7, no. 2, 73-93.

6. Naylor, T.; Balintfy, J; Burdick, D.; and Chu, K,, Computer Simulation Techniques, Wiley, New York, 1966, 43-62.

7. Gwynn, J. Persmul Communication

8. Madnick, S. and Donovan, J. Operating Systems McGraw-Hill, New York, 1974, 235-243.

9. Gwynn, J. and Pass, E. "An adaptive micro-scheduler for a multiprogrammed computer system", Proc. ACM National Conf., 327-331, 1973.

10. Needham, R. in Operating System Techniques, ed. by C.A.R. Hoare and R.H. Perott, Academic Press, London, 1972, 200.

11. Donovan, J. Systems Programming, McGraw-Hill, New York, 1972, 51-55.