LOAD VARIATION, SCHEDULING AND MEASUREMENT

W. C. Lynch

Reporters: Dr. S. Tsur
Dr. I. Mitrani
Dr. J. H. Hine

Abstract

The second of Professor Lynch's lectures was devoted to the topic of computer scheduling. He pointed out at the very beginning of his talk, that there is as much misinformation today in the area of scheduling as there was ten years ago in the area of compilers and language design.

Introduction

In general, it can be said that the aim of scheduling is to minimize some sort of "waiting time". For the purpose of this lecture, the objective is to minimize the average waiting time over all activities or, which is equivalent, to minimize the sum of the waiting times of all activities (where the waiting time of an activity is the time between its arrival into and departure from the system). Why is the total waiting time important? This is obvious as far as external users of the system are concerned - they all place some premium on their time and do not like to see it wasted. Internally, it is generally true that one module's waiting times determine another module's service time. Thus, the effects of waiting times tend to cascade through the system.

Furthermore, there is in most cases a direct relationship between waiting time and throughput, such that minimizing the average waiting time usually leads to maximizing the throughput.

Scheduling - Smallest Remaining Time First

A very simple scheduling model is one of a single-server system where all requests are available at the beginning and all service times are known in advance (see Figure 1). The solution for this case is well known - the schedule which minimizes the average wait time is the one which executes the requests in order of increasing service times. This schedule has the additional desirable property that if it is interrupted at any moment and then reapplied to the remaining jobs (provided that the service time of the current job is replaced by its remaining service time), then the resulting schedule is the same as the original one.

The above model can be generalized by allowing jobs to arrive at various times. To obtain an optimal schedule, requests should now be ranked according to their remaining execution times and the smallest-rank-first criterion should be reapplied at each arrival instant. This may lead to the preemption of the currently executed request if its remaining execution time is greater than the one of the new arrival (Figure 2). It is implicitly assumed that the cost of preemptions is zero, an oversimplification but a necessary one, if the model is to remain tractable.

Suppose now that only the distribution of service times is known, not the service times of individual requests (Figure 3). It can be shown that if requests are ranked according to their expected remaining service times (conditioned upon the service time already received) and then are executed shortest-rank-first, with rescheduling at each arrival instant, this results in an optimal schedule. However, such a scheduling discipline may be difficult to put into operation if, as in Figure 3, the expected remaining service time is an increasing function of the service time already received. In particular, if at any moment in time the jobs in the system have equal rank (e.g. because none of them has received any service), then none of these jobs can proceed without the others (because as soon as it receives some infinitesimal amount of service, its rank increases and it has to be replaced by one of the other jobs). This means that all such jobs have to proceed in parallel, each using a fraction of the processor capacity; this discipline is called *Processor-Sharing*. In practice, a *Round-Robin* (or *Time-Slicing*) discipline is operated instead of the processor-sharing one: each of the jobs is given, in rotation, a finite quantum of service.

Thus, if the expected remaining service time is increasing, the optimal scheduling strategy becomes quite complex:

> Run the newest jobs until expected remaining service time equals that of the next oldest job (or until it completed).
>
> Processor-Share (Time-Slice) between jobs of equal rank until a new job arrives (or until they complete).

If, on the other hand, the expected remaining service time is a constant, then preemption never occurs and the optimal scheduling strategy is First-In-First-Out.

It should be pointed out that these ideas - of using the past history of the request to tell something about its future behavior - can be applied at all levels of system architecture, they can be applied to internal scheduling, disk scheduling, etc.

Roll-in/Roll-out

The next example is one which involves swapping - the roll in/roll out of interactive jobs in order to utilize resources optimally. It is assumed that the object is to minimize the time jobs spend in core unproductively (Figure 4). Clearly, there is a trade-off here: if jobs are not rolled out between interactions, then the costs of wasting storage are high; if they are swapped too often, then the costs of swapping (a job has to be kept in core until it is written out) are high. In performing the optimization, one has to assume some distribution for the time between interactions: a distribution with increasing expected remaining time was chosen, since this type of distributions reflect the real-life behavior of users.

Rolling out is not usually justified immediately after the end of a computation interval; the optimal policy waits for time τ and if by then a new computation request hasn't arrived, rolls the job out (the expected remaining inactive period is now much larger). This policy is illustrated on Figure 5. The figure also shows the cost (the average unproductive time in core) as a function of the wait-before-swapping time τ. The cost is high for small values of τ (frequent swapping), reaches minimum for a certain value τ_{min} and then rises towards a horizontal asymptote as τ increases.

In operating a policy of this sort, there is a danger of an unstable positive feedback loop developing, which causes system performance to deteriorate drastically. This happens when the operating system notices that CPU utilization is low, and thinking that the current value of τ is too large, reduces it to a point below the optimum, which leads to further drop in CPU utilization, even more frequent swapping, etc.

Disk Requests

The last model discussed was concerned with servicing requests for disk input/output. In order to make the model realistic (in real-life systems the disk arms move in only half of all I/O operations), each request is assumed to consist of a sequence of blocks, consecutively arranged on a single cylinder. The next block in the sequence (if any) will be requested only after the previous has been serviced. The distribution of the interblock times is known and its mean is small compared to the disk arm move time and the rotation time. The requests arrive in a Poisson stream and are uniformly distributed across the cylinders. The distribution of the number of blocks in each request is geometric.

Clearly, there are several design decisions which should be given careful consideration. One of these concerns the size of the physical gap betwen blocks: too small a gap would not allow the next block to be requested before the disk rotates past the block and a revolution is lost; too large a gap wastes time before the next service can begin. Another design decision is: how long should we wait after a block was serviced before moving the arm to

the cylinder corresponding to another request sequence (assuming that there is one)? Too small a delay will start the arm moving before the next block of the current request arrives (if there is a next block) and will lead to an excessive amount of arm movement; too long a delay will cause too much lost time before the next request service can begin.

A mathematical optimization can be performed in order to obtain the best values for the above parameters. The shape of the distribution of interblock times plays a crucial role in the optimization. This, again, emphasizes the necessity for thinking carefully about what measurements should be performed and what kind of statistics should be taken in any given system. The answers are not always intuitively obvious.

Professor Lynch concluded his lecture by displaying an empirical histogram of rotational delays in a PDP disk unit (Figure 6). The histogram shows that a large proportion of requests have a rotational delay of one revolution and that the average delay is substantially greater than half a revolution, thus illustrating the effect of taking bad design decisions on system performance.

ASSUMPTIONS - EXAMPLE 1

- o ALL REQUESTS AVAILABLE AT THE BEGINNING - TIME 0
- o ALL SERVICE TIMES KNOWN

SOLUTION

- o RANK EQUAL TO SERVICE TIME
- o SMALLEST RANK FIRST

Example 1 Ert = St - Et

Job	A	B	C	D
Service Time	4	2	1	3
Arrival Time	0	0	0	0

	C	B	B	D	D	D	A	A	A	A	
Time	0	1	2	3	4	5	6	7	8	9	10

Job/Rank											WAIT
A	4	4	4	4	4	4	4	3	2	1	10
B	2	2	1								3
C	1										1
D	3	3	3	3	2	1					6
											20

IF WE RESCHEDULE (E.G. AT TIME = 3), WE SHOULD USE THE REMAINING SERVICE TIME RATHER THAN THE ORIGINAL SERVICE TIME

NOTE THAT RESCHEDULING DOES NOT CHANGE THE OPTIMAL SCHEDULE

Figure 1

ASSUMPTIONS - EXAMPLE 2

- o REQUESTS ARRIVE AT VARIOUS TIMES
- o ALL SERVICE TIMES KNOWN

SOLUTION

- o RANK EQUAL TO REMAINING SERVICE TIME
- o SMALLEST RANK ALWAYS FIRST
- o PRE-EMPT IF NECESSARY
- o RESCHEDULE ON AN ARRIVAL

Example 2 Ert = St - Et

Job	A	B	C	D
Service Time	4	2	1	3
Arrival Time	0	0	3	0

	B	B	D	C	D	D	A	A	A	A	
Time	0	1	2	3	4	5	6	7	8	9	10

Job/Rank											WAIT
A	4	4	4	4	4	4	4	3	2	1	10
B	2	1									2
C				1							1
D	3	3	3	2	2	1					6
											19

Figure 2

ASSUMPTIONS - EXAMPLES 3-4

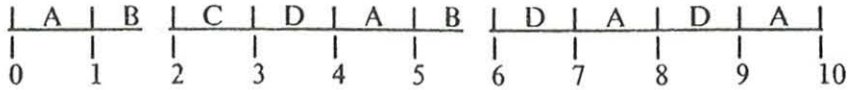
- o REQUESTS ARRIVE AT VARIOUS TIMES
- o ONLY THE SERVICE TIME DISTRIBUTION IS KNOWN
- o PROB(SERVICE TIME > t) = $4/(2+t)^2$
- o EXPECTED REMAINING TIME AFTER τ IS
  $E(t-\tau \mid t > \tau) = 2+\tau = E(\tau)$

SOLUTION

- o RANK EQUAL TO EXPECTED REMAINING SERVICE TIME $E(\tau)$
- o SMALLEST RANK ALWAYS FIRST
- o RESCHEDULE ON AN ARRIVAL

Example 3 Ert = 2 + Et

Job	A	B	C	D
Service Time	4	2	1	3
Arrival Time	0	0	0	0

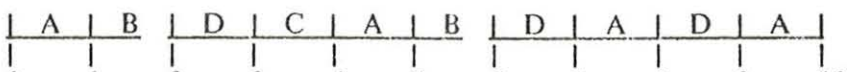


Time	0	1	2	3	4	5	6	7	8	9	10

Job/Rank											WAIT
A	2	3	3	3	3	4	4	4	5	5	10
B	2	2	3	3	3	3					6
C	2	2	2								3
D	2	2	2	2	3	3	3	4	4		9
											28

Example 4 Ert = 2 + Et

Job	A	B	C	D
Service Time	4	2	1	3
Arrival Time	0	0	3	0



Time	0	1	2	3	4	5	6	7	8	9	10

Job/Rank											WAIT
A	2	3	3	3	3	4	4	4	5	5	10
B	2	2	3	3	3	3					6
C				2							1
D	2	2	2	3	3	3	3	4	4		9
											26

IF THE EXPECTED REMAINING TIME IS INCREASING

o RUN THE NEWEST JOB UNTIL THE REMAINING TIME EQUALS THAT OF THE NEXT OLDEST

o PROCESSOR SHARE (TIME SLICE) BETWEEN THESE TWO

o PUSH JOBS INTO THE BACKGROUND AND RUN NEW ARRIVALS IN THE FOREGROUND

Figure 3

o SWAPPING - ROLL OUT JOB UNTIL NEXT INTERACTIVE INPUT

o OBJECT - MINIMIZE UNPRODUCTIVE TIME IN CORE

  o IF NO SWAP, THEN WHOLE TIME TO NEXT INTERACTION

  o IF SWAP, THEN TIME τ TO SWAP DESCISION PLUS SWAPPING TIME

ASSUME TIME TO NEXT INTERACTION HAS INCREASING EXPECTED REMAINING TIME

o WE USE AS AN EXAMPLE

  PROB(INTERACTION TIME > t) = $4/(2 + t)^2$ = P(t)

  $E(t-\tau \mid t > \tau) = 2 + \tau = E(\tau)$

SOLUTION

  o WAIT τ INTO INTERACTION AND THEN SWAP OUT IF INTERACTION IS NOT ENDED

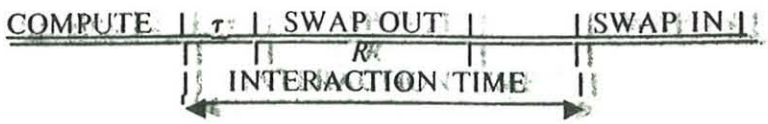
  o WHEN INTERACTION IS ENDED THEN SWAP IN

  o SWAP OUT AND SWAP IN TIMES ARE EACH R

Figure 4

CASE 1 - SHORT INTERACTION TIME LESS THAN τ

CASE 2 - LONG INTERACTION TIME



UNPRODUCTIVE TIME IN CORE AS A FUNCTION OF τ IS

$$E(0) + P(\tau)(2R - E(\tau))$$

E.G. $\quad 2 + 8R/(2+\tau)^2 - 4/(2+\tau)$

Figure 5

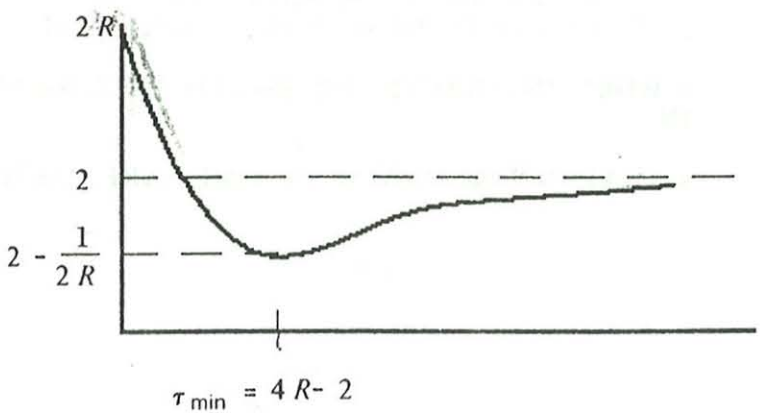


Figure 5

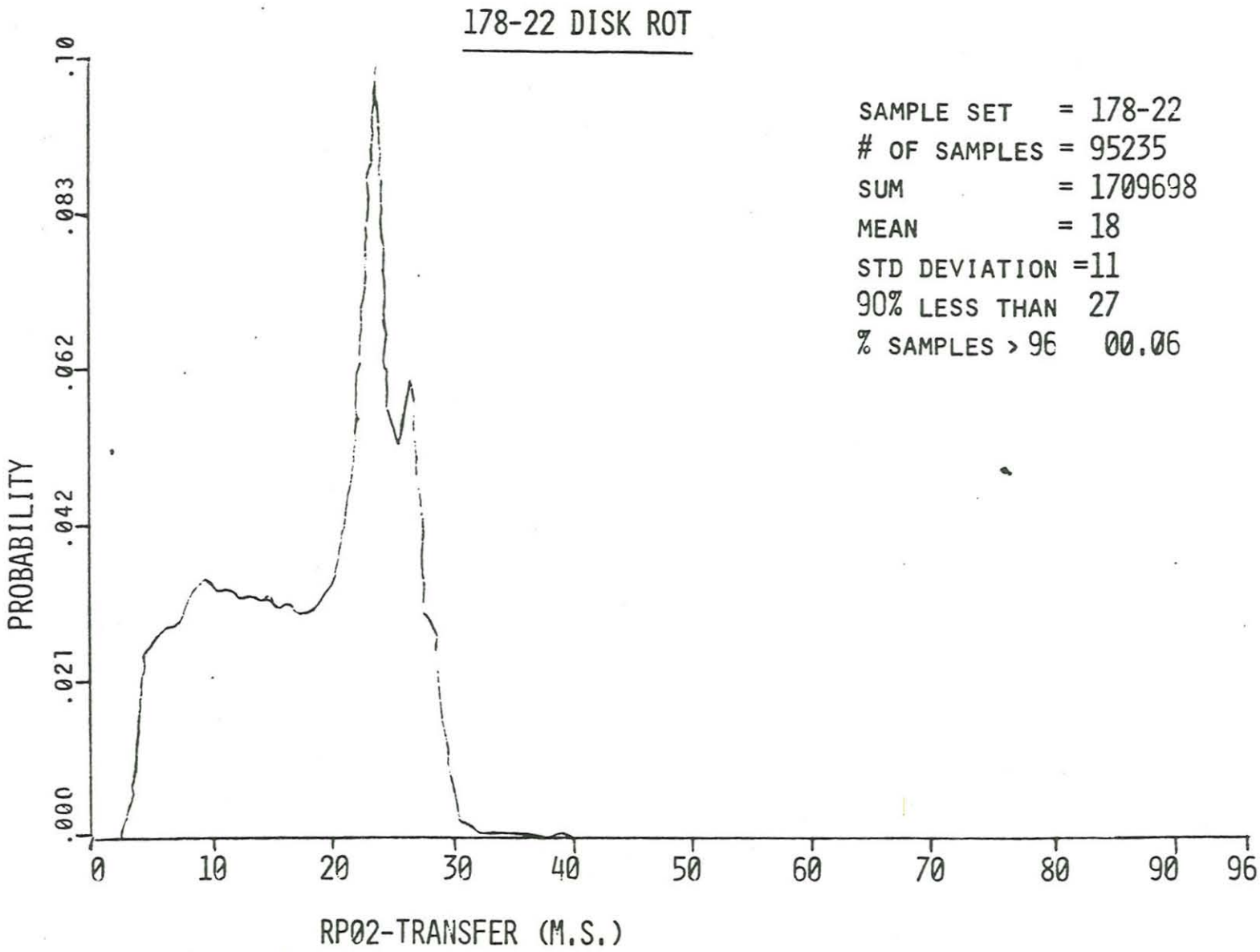

178-22 DISK ROT
SAMPLE SET = 178-22
# OF SAMPLES = 95235
SUM = 1709698
MEAN = 18
STD DEVIATION = 11
90% LESS THAN 27
% SAMPLES > 96 00.06
PROBABILITY
.000
.021
.042
.062
.083
.10
0
10
20
30
40
50
60
70
80
90
96
RP02-TRANSFER (M.S.)

Figure 6