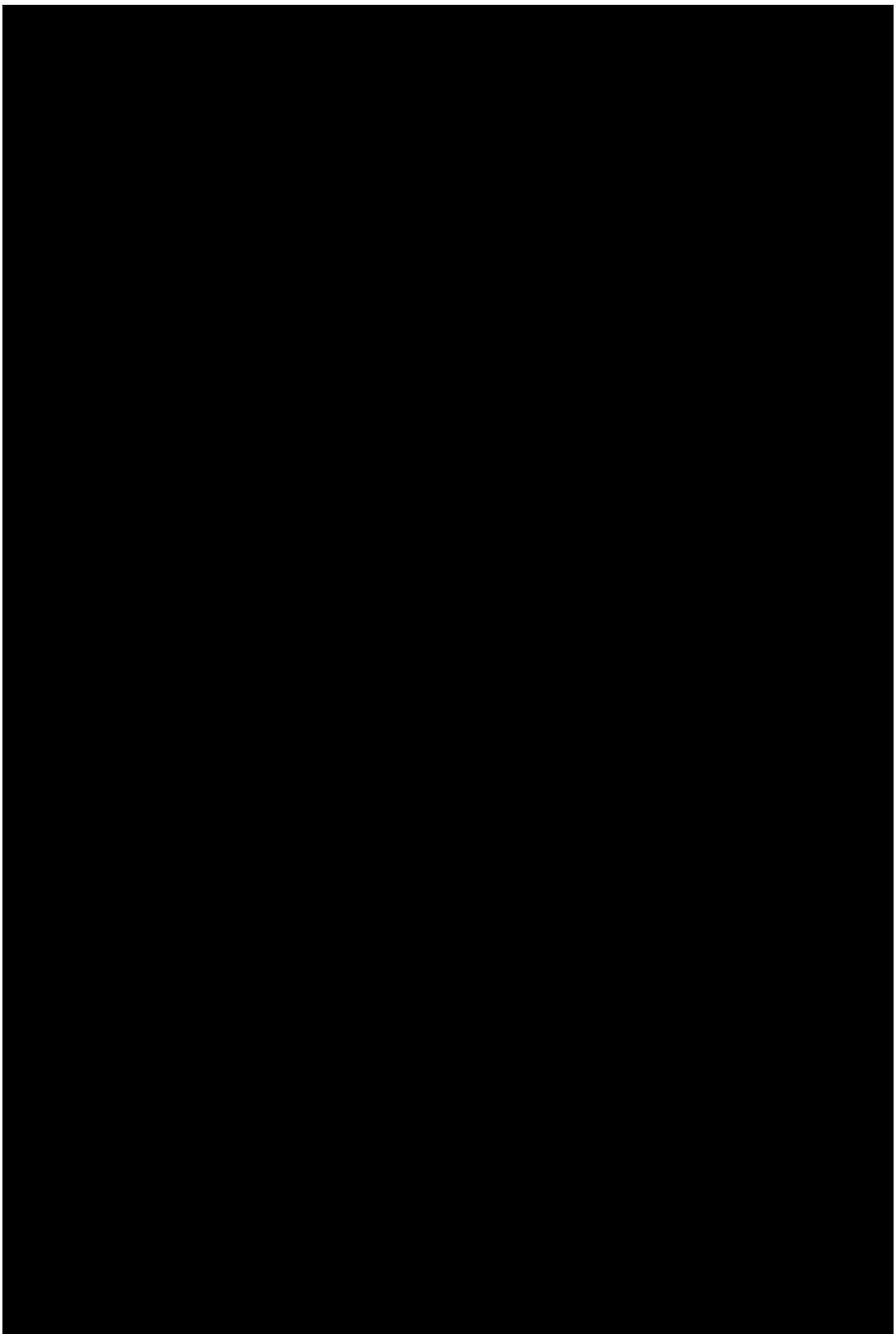


This blank page was inserted to preserve pagination.

ADDENDUM

Development of the analytical model for time sharing and calculation of the results reported in Figures 4 and 5 were facilitated by the OPS on-line computer system. This system is described in the book On-Line Computation and Simulation: The OPS-3 System, by Martin Greenberger, Malcolm M. Jones, James H. Morris, Jr., and David N. Ness, MIT Press, 1965.

This empty page was substituted for a blank page in the original document.

MAC-TR-22

Massachusetts Institute of Technology

Project MAC

and the

Sloan School of Management

THE PRIORITY PROBLEM

by

Martin Greenberger

November 1965

Portions of this paper were presented in a talk at the 27th National Meeting of the Operations Research Society of America in Boston on May 6, 1965.

This empty page was substituted for a blank page in the original document.

ABSTRACT

Priority decisions arise whenever limited facilities must be apportioned among competitive demands for service. Broadly viewed, even the familiar first-come-first-served discipline is a priority rule. It favors the longest-waiting user, and guards against excessive delays. Other priority rules, such as shortest-job-next, are keyed instead to considerations of operating efficiency. Urgency of request is still another common consideration. Since these considerations often conflict, the priority rule serves as mediator. Use of a common cost measure can help effect this mediation, as results from recent job-shop simulations illustrate.

A priority operation of contemporary interest is scheduling a time-shared computer among its concurrent users. Service requirements are not known in advance of execution. To keep response times short for small requests, service intervals are partitioned and segments are served separately in round-robin fashion. A mathematical analysis pinpoints the tradeoff between overhead and discrimination implicit in this procedure, and allows alternate strategies to be costed. Extensions of the simple round-robin procedure are suggested, the objectives of time sharing are reviewed, and implications are drawn for the design of future priority and pricing systems.

ACKNOWLEDGEMENTS

The author's interest in the priority problem was stimulated by a 1960 summer study on problems in Naval Command, Control, and Communications, and by informal discussions with E. H. Bowman, P. M. Morse, other colleagues who participated in that study, and J. D. C. Little. Also enjoyable and helpful were more recent conversations with students and co-workers at Project MAC including M. M. Jones, N. Patel, M. Rothkopf, A. L. Scherr, J. H. Saltzer, and M. Wantman.

TABLE OF CONTENTS

Section	Page
ABSTRACT	i
ACKNOWLEDGEMENTS	ii
LIST OF ILLUSTRATIONS	iv
INTRODUCTION	1
CONFLICTING OBJECTIVES	2
COST RATE CURVES	3
THE c/t RULE	5
JOB SHOPS AND DEADLINES	8
THE COMPUTATION CENTER	10
TIME SHARING	11
ROUND-ROBIN SCHEDULING	13
AN ANALYTICAL MODEL	16
COSTING THE MODEL	20
MULTIPLE LEVELS AND VARIABLE QUANTA	24
CONCLUDING REMARKS	26
REFERENCES	28

LIST OF ILLUSTRATIONS

Figures		Page
1	Examples of Cost Curves	4
2	Nonlinear Costs	6
3	Service Completions Under Three Priority Rules	14
4	Cost Performances as a Function of Quantum Size (N and γ parametric)	21
5	Cost Performances as a Function of Quantum Size (N and γ fixed)	23

INTRODUCTION

Granting preferred treatment is a basic part of man's way of life, democratic principles notwithstanding. Whenever demand temporarily exceeds supply, or a limited facility must be rationed, or only one of several can be served at a time, someone accepted means someone else refused or deferred. Implicitly or explicitly, a question is being posed and resolved. In formal terms, we refer to the question as a priority problem and its solution as a priority decision. A strategy or objective plan for routinely making priority decisions, we call a priority rule.

The priority problem arises in a variety of contexts. In a supermarket, customers with small orders are ushered to a special check-out counter. This expedites their service, and has the desirable secondary benefit of reducing congestion in the store. Giving priority to small requests is often a very good idea, but not always easy, as later discussions will illustrate.

Importance or urgency is a second factor entering into priority decisions. At an airport with a single runway that is used jointly (but not simultaneously) by arriving and departing aircraft, a landing plane may be given precedence over one waiting to take off, even though the departing plane was the first to request permission from the tower. This is especially true for an arriving jet whose tight budget of fuel is close to exhaustion.

A priority rule comes closest to being democratic in the familiar first-come-first-served or first-in first-out dicipline. But even this policy is preferential, in the sense that it systematically favors the customer who has been waiting the longest. In doing so, it safeguards against excessive waits and controls the variance of waiting.

CONFLICTING OBJECTIVES

There are at least three different, and usually conflicting objectives that can influence a priority decision and affect the design of a priority rule:

1. Reduce average response time and number waiting; thus increasing throughput or rate of output,
2. Acknowledge customer importance and urgency of request,
3. Serve in fair order and limit length of wait.

For the best average performance, the shortest-service-time-next rule may be just right. But this rule allows a steady stream of short requests to delay a long request indefinitely. The mean wait is minimized at the expense of the variance, and the special interests of the long user are sacrificed for the general welfare. To accord special interests their due, a balance must be struck among conflicting objectives. This requires a common measure of performance that incorporates the interests of the individual. Any contrived measure is going to be susceptible to challenge, but that is healthier than grumblings about seeming arbitrariness.

COST RATE CURVES

We choose an inverse measure of performance: the cost of delay. Depending on the context, it may be thought of as a disutility, penalty cost, loss of goodwill, opportunity cost, postponement of revenue, customer dissatisfaction, storage cost, poorness of service, or some equivalent. We first used the cost measure several years ago during a study of message priority in a communication system[7]. Each message (or customer) type was considered to have a separate cost rate curve, as illustrated in Figure 1. These curves plot rate-of-accrual of cost versus time of waiting. Curve (a) shows the simplest case of cost accruing at a constant rate, c, throughout the period that a message is waiting to be transmitted, coded, decoded, or, in general, served. If the message waits a time w, the total cost added to the books of the operation is cw. This is the case of linear costs. An order to replace an aircraft engine for a grounded plane might have such a curve.

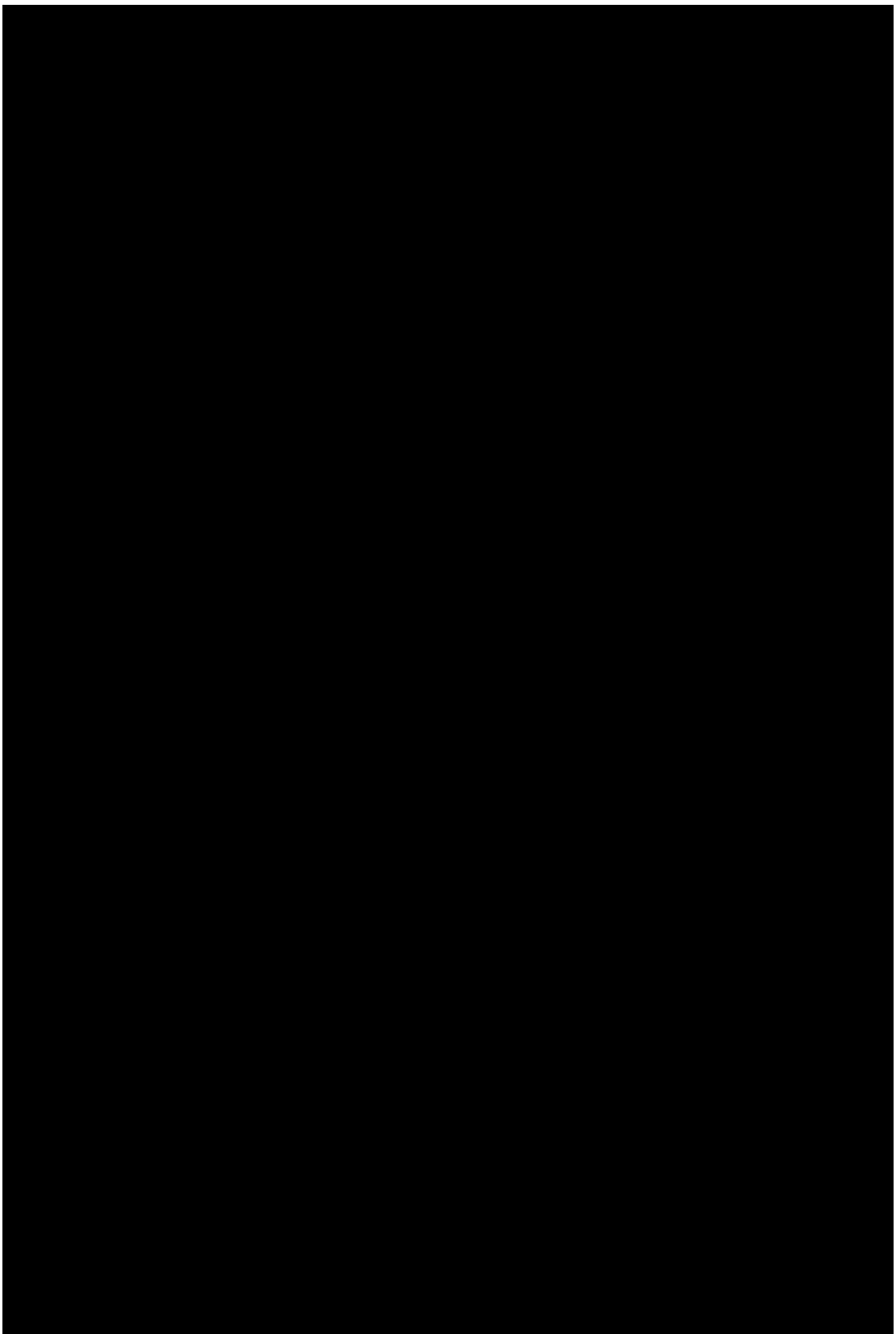
Curve (b) depicts a quadratic cost case. The longer a message waits, the greater its marginal cost. An important command from higher headquarters might have either this characteristic or the exponential variations of it portrayed by curves (c) and (d).

Curve (e) illustrates a message whose quick transmission is important, but whose timeliness decays exponentially. Examples might include the warning of a missile attack, or, on a different scale, a weather forecast.

Curve (f) is an artist's rendition of what a general cost curve looks like. Note that in each of these examples the integral of the curve over the duration of wait equals the total cost attributable to the message.

Assignment of cost rate curves to customers makes the conflicting objectives commensurable by reducing them to a common measure of system performance. The importance of a customer is reflected by the height of his curve. His aversion or intolerance to delay is represented by the shape of his curve. Minimizing the total cost accrued by all customers becomes the single system objective.

If all customers have identical cost rate curves, and if this curve is constant as in the first case of Figure 1, then minimizing cost accrual is equivalent to minimizing average wait. Thus, in this very simplest case, introduction of the cost measure does not complicate the traditional analysis. We shall see, however, that the complexity of the analysis rises sharply with even small complications to the cost measure.



THE c/t RULE

The next simplest case is that of constant cost-rate curves, where the constants are different for different customers. In a classical single-server situation, this leads to what has been called the $c\mu$ or c/t rule. That is, if c_i is the cost rate of the i^{th} unit waiting for service, and t_i is its expected service time ($\mu_i = 1/t_i$ is its service rate), then the unit with highest c_i/t_i should be served next. When all c_i are equal, servicing in ascending order of t_i is optimal and the c/t rule reduces to the rule of shortest-service-time-next. References on the c/t rule are available in many places[1,4,12].

Note that the dimensions of c/t are cost/(time^2). Thus, the c/t of a unit is an acceleration that indicates how quickly the cost accrual velocity could be reduced by serving that unit next. Past history is not relevant to the priority decision in the linear case; nor is there interaction among units. Unfortunately, these simplifications do not apply to the general nonlinear case.

To examine the nonlinear case in a simple situation, suppose that at time t_0 a choice must be made between two units waiting for service. The first unit has a service requirement of t_1 and the second unit of t_2. Assume that during the interval $(t_0, t_0+t_1+t_2)$ both of these units are served and no other units arrive.

The cost rate curves of the two units are given in Figure 2. If the first unit receives priority, the cost accumulated during the interval is

$$\int_{t_0}^{t_0+t_1} c_1(t)dt \quad + \quad \int_{t_0}^{t_0+t_1+t_2} c_2(t)dt \tag{1}$$

whereas, if the second unit receives priority, it is

$$\int_{t_0}^{t_0+t_1+t_2} c_1(t)dt \quad + \quad \int_{t_0}^{t_0+t_2} c_2(t)dt \tag{2}$$

The first unit should be given priority if and only if expression (2) is greater than or equal to expression (1). That is,

$$\int_{t_0+t_2}^{t_0+t_1+t_2} c_2(t)dt \quad \leq \quad \int_{t_0+t_1}^{t_0+t_1+t_2} c_1(t)dt \tag{3}$$

If we let $\bar{c}_1$ be the average value of $c_1(t)$ during a period of duration t_2 beginning at t_0+t_1, and $\bar{c}_2$ be the average value of $c_2(t)$ during a period of duration

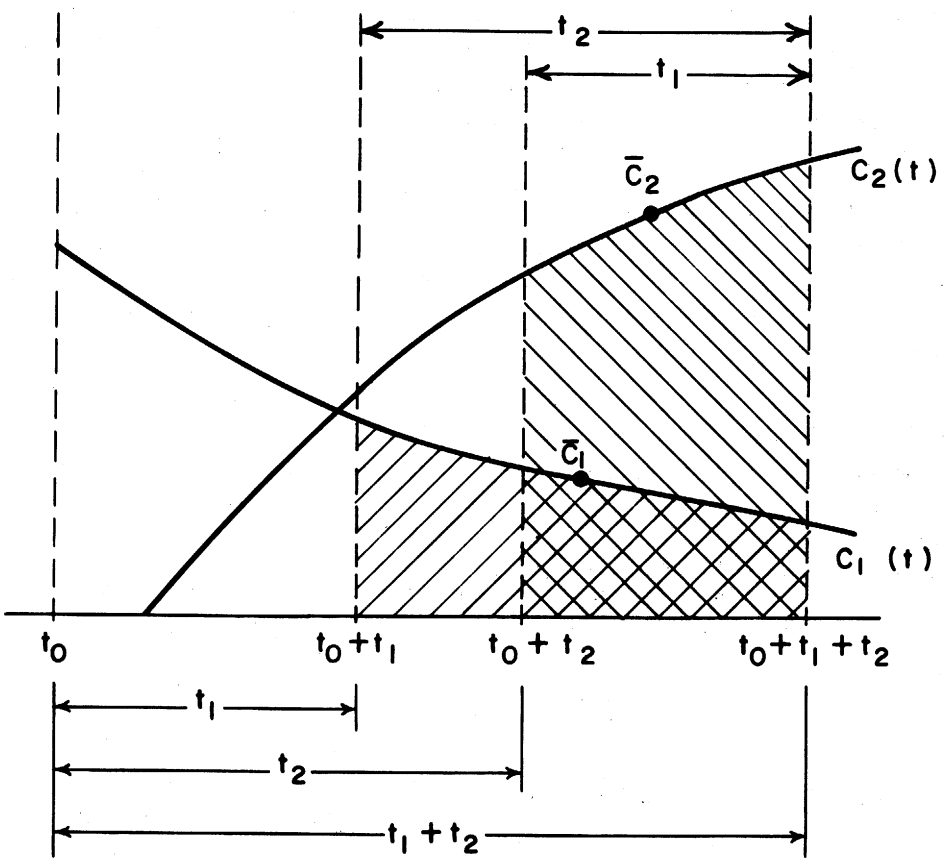
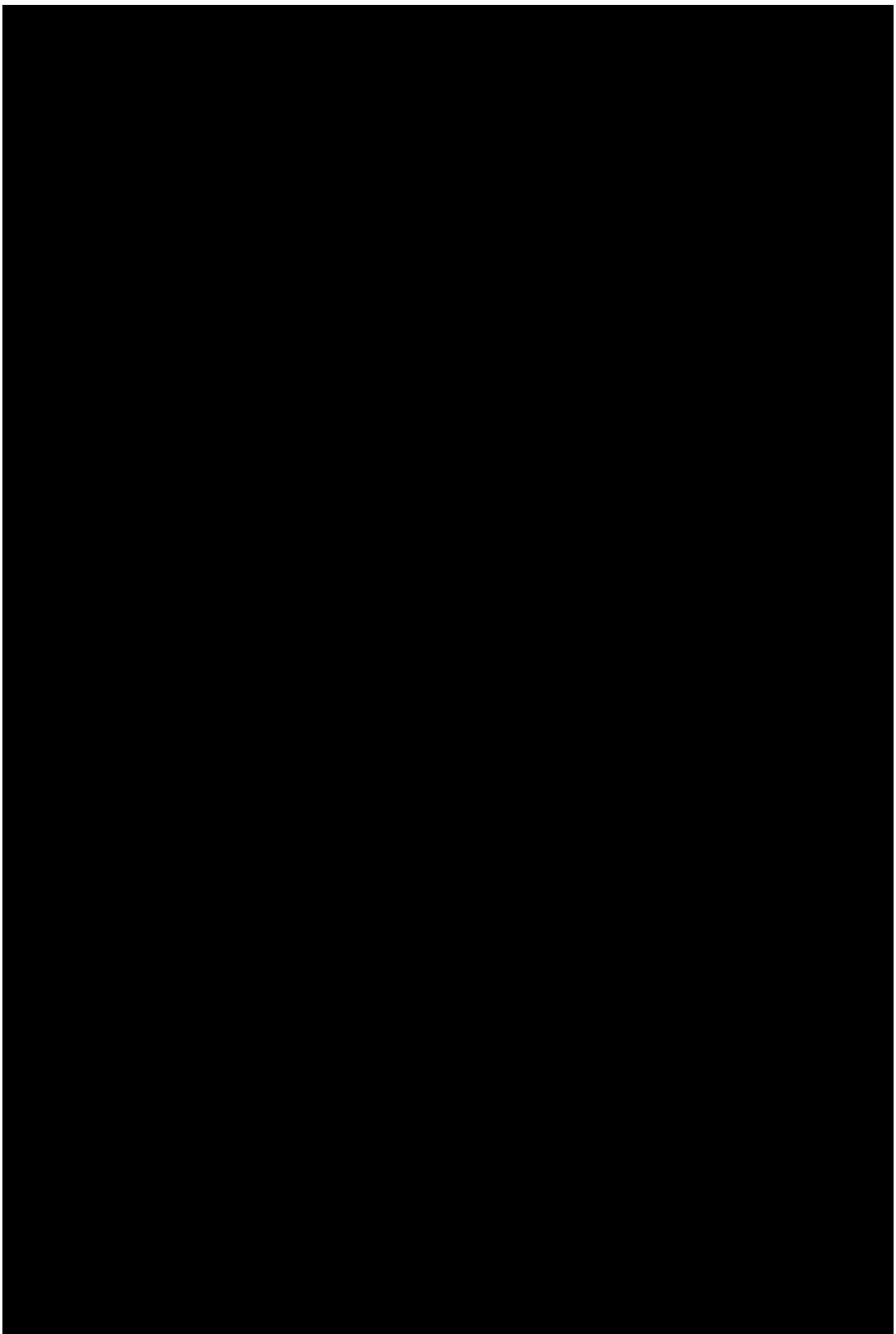


Figure 2. Nonlinear Costs



JOB SHOPS AND DEADLINES

The priority problem is found in too many places for us to make an exhaustive list. All of us face priority problems every day of our lives, no matter what our occupation. We shall focus on two areas where the priority problem assumes central importance: job shops and computers.

Work on job-shop priorities has been going on for a number of years and several good surveys exists. Carroll, for example, gives a comprehensive overview of the field and presents some interesting new results that illustrate the value of costing[1].

In a job shop, jobs are typically promised by a certain date called the due date or deadline. Each job consists of a number of tasks to be performed on different machines. Call this number N. Since a job must be served N separate times, it is subject to N successive priority decisions: when it has N tasks left to accomplish, when it has N-1 tasks left, and so on, until it has only one task left.

Suppose that a pair of tasks from two different jobs are competing for service at a machine. The first task requires less machine time than the second, but the second job is in greater danger of missing its deadline. Which job should be given priority?

Giving priority to the first job contributes to the objectives of high throughput and low average wait, but may cause a default on the second job's deadline. Giving priority to the second job has the opposite effect. This is the classical trade-off. Carroll's results indicate that cost curves can help in striking a balance.

Carroll postulates a cost rate equal to the probability of a job's being late. In his model, this is the rate of change of the job's expected amount of lateness. The rate applies to the task of the job that is currently awaiting service. It increases monotonically toward a value of one, as the deadline of the job is approached, then remains at one from the deadline until the last task of the job has been completed.

The conceptual model is straightforward, but the probability of being late is difficult to estimate accurately. Even if precise machine times for all tasks are known with certainty, waits between task initiations can only be approximated. The approximations should be based on current loads and revised as loads shift. Moreover, estimated probability of being late for a job should depend on the number of its remaining tasks, as well as on the approximated distribution of wait for each of these tasks.

Carroll sets aside such refinements for later study, and chooses a simple expedient. He estimates the sum of expected waits over the remaining tasks, and uses this estimate to postulate a threshold time beyond which he assumes the probability of being late rises linearly from zero to one at the deadline. With this probability as c, he invokes the c/t rule at each decision point. Theoretically, c should not vary, but in Carroll's model it is changing all the time. The end result of this compounding of assumption upon simplification upon approximation is a rule that produces consistently fewer late jobs in simulations than any previously simulated rule.

The implications of this work extend beyond the scheduling of job shops. As one example, there is a decided trend today toward greater complexity in the organization of computer systems. It is a safe guess that future computer systems will consist of many processors, many separate memory modules, many input-output terminals and data coordinators, with flexible interconnections, simultaneous users, and concurrent operation. Lessons learned from the study of job shops will carry over to questions of scheduling space, time, and program access in these computer systems of the future.

THE COMPUTATION CENTER

The computer field is beginning to show greater interest in the priority problem as machine structures grow more complex, but some concern with the problem has existed in the field from the very start. Historically, as soon as a computation center had two users, it also had priority decisions to make. Most open-shop centers provided an informal solution: sign-up sheets. First on the sheet was first on the machine, or at least had first choice.

Other centers used more elaborate procedures. The IBM 701 Scientific Computing Service in New York City, circa 1954, had a dispatcher who sat on a glass-enclosed balcony overlooking the 701 computer. One or more eager customers sat alongside her, ready to pounce whenever their predecessor on the machine signalled completion (or frustration) and punched out on the IBM time clock at the console. The dispatcher made certain that the queue on the balcony was never empty by phoning users in their offices and alerting them well ahead of time. A carefully-designed priority formula allowed certain customers, like Los Alamos, to gain access to the computer on very short notice.

Even this did not prevent momentous inefficiency and idle time. As computers became more advanced and more expensive, inefficiency had to be wrung out of the operation. The larger computation centers became closed-shop, and jobs were batched serially on tape before run time to provide fast transition between successive executions.

At heavily-loaded centers, such as the M.I.T. Computation Center in 1958, turnaround times soared to several days, and sometimes to a week or more, despite batching and the use of professional operators. Priority rules gave some relief to special users. Urgent needs, such as the Sputnik orbital calculations (which provided settings for camera and telescope stations around the world) were given preemptive rights. Other business ceased when a satellite was launched. In normal periods, short jobs were awarded express service at prescribed times of day, and very long jobs were deferred to night-time shifts.

TIME SHARING

Expediting service for the short user just whetted the appetite for more frequent access. Freeing a modest-sized program of errors by means of the customary iteration of running, modifying, running, and modifying, could require weeks or months in a batch-processed operation, as compared to an afternoon or evening spent at a computer (such as M.I.T.'s TX-0 or TX-2) which the programmer had to himself for a while. A private computer not only saved the user time, it also allowed him to interact with his program, view preliminary results, and alter experimental strategy on-line[10].

The recent development of time-sharing gives the impression and advantages of a private computer to simultaneous users at remote consoles of a large computer[3,5,8]. This development is a concession to the user and recognizes his point of view. Overall system objectives suffer initially, but equipment being built with new system concepts will restore and ultimately improve past levels of operating performance.

If we view the purpose of time sharing as the creation of a privileged class of user for whom the computer is continually accessible and immediately responsive, then, in the spirit of traditional express service, we are led naturally to establish the short request as the privileged user. It is impossible, by definition, to serve a long request instantly. Moreover, to the extent that the computer does give precedence to long requests, its responsiveness to short requests is degraded and the purpose of time sharing is undermined.

Thus, we tend to favor the class of requests for whom we can offer fast service. This produces an unusual situation. User and system objectives are of a single mind in that both direct us to favor the short request, assuming that we can identify it ahead of time. Unfortunately, we generally cannot. It is only after execution that we really know which was the short request and which was the long request.

The typical time sharer sits at his console for an hour or more issuing requests, being served, making inquiries, and receiving replies. We do not want to ask him before each interaction how much time he expects to take for two reasons. First, we could not place much faith in his response, because the mind consciously or subconsciously tends to underestimate requirements. Second, time sharing is at its best when the computer and its characteristics are inconspicuous. The user should not have to be aware of his consumption of computer time for each step he takes.

Can the computer anticipate time requirements without being told? Yes, to some extent; especially when a request is a standard command or program which the computer knows by name. A scheduling based on such information would be context-sensitive, to borrow a term. M.I.T.'s time-sharing supervisor infers the size of a program from its name, but does not make inferences about its expected time of execution[13]. For the following mathematical discussion, we shall adopt this conservative position of assuming no prior knowledge of request time.

ROUND-ROBIN SCHEDULING

Discriminating against long requests, when we are able to identify which ones are long, prevents them from delaying short requests. When we cannot identify them, we hedge. We serve a request for some fixed amount of time, called a quantum. If that is not sufficient, we suspend its service, place the balance of the request at the end of the queue, and go on to the next request. If the quantum is sufficient, we give the unit only enough time to complete its service. This types of a priority rule has been called round-robin scheduling[10].

Under round-robin scheduling, the longer request is split or partitioned during execution, and its segments served at separate times. In general, there are two instances when it may pay to partition jobs in a service operation: when there is more than one server, and when there is uncertainty in service times[12]. Only the second case applies here, since we shall be assuming a single processor, but both cases are relevant to future time-sharing systems with multiple processors.

To illustrate how partitioning can be helpful under conditions of uncertainty, suppose that half of all requests are for 1 second of computer time, and the other half for 10 seconds. The computer does not know which is the 10-second request before execution, but by partitioning with a quantum of 1 second it acquires this information at a charge of 1 second, plus an additional overhead charge of V occasioned by suspension of service on the incomplete request. Thus a 1-second request arriving for service immediately after a 10-second request is detained by its predecessor 1+V seconds rather than 10 seconds. The 1+V charge is like a cost of information, although its 1-second component does substract from the 10-second request. Partitioning contributes to the system objective of lower average wait, as well as to the user objective of better response time for the short request, giving a double advantage in this example.

Now we consider an example where partitioning does not even offer a single advantage. Suppose that all requests are known in advance to be for 10 seconds of computing time. Round-robin scheduling with a quantum of 1 second is then absurd. It introduces unnecessary delays without providing any new information. Round-robin scheduling with a quantum of 10 seconds (or greater) makes better sense, but is nothing more than a first-come-first-served rule.

These observations lead us to the first of a few informal points that will be made without proof or detailed discussion.

Point 1 The benefits of round-robin scheduling (RRS) relative to first-come-first-served (FCFS) increase with the uncertainty of request sizes. The variance of the distribution of request sizes may be taken as a measure of this uncertainty.

In general, we consider both request size and the user's think time at his remote console as random variables. The think time is defined to be the interval between completion of a user's request and initiation or arrival at the processor of his next request. The time-shared operation is characterized by periods during which one request is in service and others may be in queue, called busy periods, and periods during which no request is either in service or in queue, called idle periods. Idle periods may be utilized by the processor to nibble at deferred work stored in reserve.

We now consider another situation. Suppose that an idle period has just been terminated by the arrival, in quick succession, of four requests for service. The first request is for four quanta of time, the second for two quanta, the third for three quanta, and the fourth for one quantum. If the requests were served in their entirety in strict first-come-first-served order, the service intervals and service completions would be as in Figure 3a. The scales shown are in quantum units, and overhead is assumed to be negligible. If it were possible to serve the shortest job next, the service completions would be, instead, as in Figure 3b, whereas under round-robin scheduling, they would be as in Figure 3c.

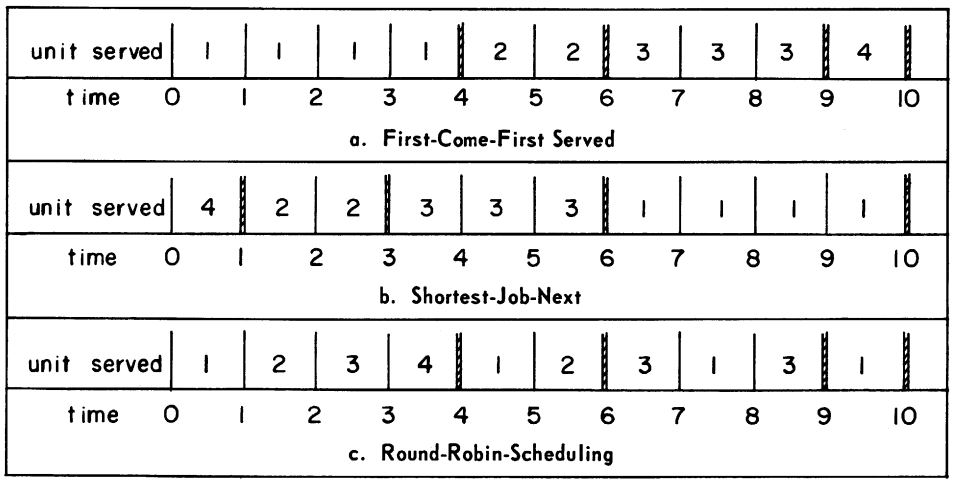


Figure 3. Service Completions Under Three Priority Rules

Notice the differences. The total time spent in waiting and service under the first-come-first-served rule is

$$W_{FCFS} = 4 + 6 + 9 + 10 = 29,$$

and the shortest request is completed at the end of quantum 10. Under the shortest-job-next rule

$$W_{SJN} = 1 + 3 + 6 + 10 = 20,$$

and the shortest request is completed at the end of quantum 1. Under round-robin scheduling

$$W_{RR} = W_{FCFS} = 29,$$

and the shortest request is completed at the end of quantum 4. Observe that the RR and FCFS rules have identical schedules of service completions. The identity is coincidental, although it suggests two general points, namely:

Point 2 When the sizes of service requests are exponentially distributed, spacings between service completions in the busy period under round-robin scheduling are also exponentially distributed with the same mean. The service completions under RRS may be thought of as reordered FCFS service completions, with completions of the shorter requests moving forward and those of the longer requests moving backward. Average throughput is unchanged if overhead is ignored.

Point 3 When the sizes of service requests are exponentially distributed, the balances of these requests in excess of the quantum size are also exponentially distributed with the same mean. The same is true of the balances, and so on. The segments of partitioned requests from an exponential distribution all share a common truncated exponential distribution, regardless of their positions in the partitioning.

AN ANALYTICAL MODEL

Points 2 and 3 provide the theoretical groundwork for an analytical model of round-robin scheduling. Let there be N consoles connected to a single time shared processor. Requests for service are assumed indistinguishable between consoles and exponentially distributed with mean $1/\sigma$ and density function

$$f(R) = \sigma e^{-\sigma R} \qquad R \geq 0 \tag{9}$$

The scheduler employs a quantum of size Q which, by Point 2, implies a cumulative distribution of segment sizes equal to

$$F(S) = 1-e^{-\sigma S} \qquad Q > S \geq 0 \tag{10}$$

$$= 1 \qquad S \geq Q$$

Requests for service are assumed to arrive at the processor from thinking consoles randomly, at a rate α per console. Thus, if J is the number of consoles waiting for service at some arbitrary time t, $(N - J)\alpha$ is the arrival rate of requests at t. Thinking times are assumed indistinguishable between consoles and exponentially distributed with mean $1/\alpha$ and density function

$$f(T) = \alpha e^{-\alpha T} \qquad T \geq 0 \tag{11}$$

The assumption of exponentially-distributed think and request times holds up well in a time-sharing operation where there is wide diversity of users, as there is at M.I.T.

The formulation permits us to employ the queuing model for machine servicing (or interference) which was developed in 1933 and has had considerable practical application since then[4,6]. We draw an analogy between time-sharing and machine servicing by viewing consoles in the role of machines, and the computer processor in the role of a repairman who services the machines. A console thinking is like a machine working, and a console waiting is like an idle machine in need of repairs. The machine-servicing model extends easily to the case of several repairmen, and is therefore applicable to a time-sharing system of multiple processors. This extension would be useful for further development of the present work, and is employed in the doctoral dissertation by Scherr[14].

An equivalent, but more subtle analogy exists between time-sharing and calls coming into a telephone exchange containing N trunk lines and no facility for holding. The arrival and server processes are interchanged in this analogy. Waiting consoles become open trunk lines; thinking consoles become busy trunk lines; the arrival of a request for computer time becomes

termination of a telephone call; and completion of a computer request becomes arrival of a telephone call when one or more lines are open. Calls that arrive when all lines are busy do not enter the exchange, and are lost. This model has been widely applied in the communications field and produces the well-known loss formula attributed to Erlang. The finiteness of trunk lines, like the finiteness of machines in machine servicing and the finiteness of consoles in time-sharing, is a distinguishing characteristic of the model.

To apply the machine-servicing model to round-robin scheduling of a time-shared computer, we must incorporate partitioning of service requests and introduce overhead. Partitioning is represented by f(S), the density function of segment sizes, whose first two moments are

$$S_1 = \frac{1}{\sigma}(1 - e^{-\sigma Q}) \tag{12}$$

$$S_2 = \frac{2}{\sigma}(S_1 - Qe^{-\sigma Q}) \tag{13}$$

Overhead is accounted for by adding the constant V to S_1, and $(2S_1V + V^2)$ to S_2. We may think of V as an average time required to bring a program into and out of primary storage. The effect of V is to lengthen request sizes and degrade the rate of processing. The smaller that quantum size Q is for a given R, or the larger that request size R is for a given Q, the greater number of segments into which R is partitioned and the more R is prolonged by overhead. The mean of f(R'), the distribution of prolonged request sizes, is given by

$$1/\sigma' = 1/\sigma + V/(1-e^{-\sigma Q}) \tag{14}$$

In practice, f(R) is only approximately exponential, and extending the exponential assumption to include f(R') may not weaken the approximation significantly. We take this liberty, even though there do exist non-exponential formulations for the machine-servicing problem[15]. Using σ' in the simple exponential formulation, and letting P_J be the steady-state probability that J of the N consoles are waiting, we get

$$P_J = \left(\frac{(\sigma'/\alpha)^{N-J}}{(N-J)!} \right) \div \left(\sum_{I=0}^{N} \frac{(\sigma'/\alpha)^{N-I}}{(N-I)!} \right) J=0,1,\ldots,N \tag{15}$$

P_J is a truncated Poisson probability. J=0 produces Erlang's loss formula.

When a request arrives at the processor from a console that has just passed from thinking to waiting, an interval may elapse before its first segment begins receiving service. Call the expected value of this interval, Y_1, its first cycle time. In the same manner, let Y_i be the expected value of the interval from the time its (i-1) segment is completed until the time its i

segment is initiated, for $i \geq 2$. Notice that the same Y_i is common to all requests containing i or more segments.

To obtain an expression for Y_i, we modify the reasoning of Cobham[2] to take account of partitioning and the state-dependent arrival rate. The probability that there are J consoles waiting (and in service) immediately prior to an arrival is

$$P(J \mid \text{arrival}) = P_{J+1}/(1-P_0) \qquad J=0,1,\ldots,N-1 \tag{16}$$

This is also the probability that there are J+1 consoles waiting, given that the system is busy, which implies that the expected number of consoles waiting, immediately prior to an arrival, and the expected number of consoles in line (exclusive of the possible one in service) when the system is busy, are equal to each other and to

$$L_B = N/(1-P_0) - \sigma'/\alpha - 1 \tag{17}$$

The expected number of consoles in line (exclusive of the possible one in service) immediately prior to an arrival is

$$L_1 = L_B - \frac{1-P_0-P_1}{1-P_0} \tag{18}$$

The expected time to completion of the possible segment in service, given an arrival, is

$$Y_0 = \left(\frac{S_2+2S_1V+V^2}{2(S_1+V)} \right) x \left(\frac{1-P_0-P_1}{1-P_0} \right) \tag{19}$$

An expression for Y_1 is

$$Y_1 = Y_0 + L_1 (S_1+V) \tag{20}$$

which is the expected time to finish the segment in service, plus the expected time to serve the segments in line ahead of the new arrival.

Let us tag the new arrival and follow its progress through the system. After its first segment has completed service, assuming there is a positive balance remaining, the balance returns to the end of the line. Call the expected number of segments ahead of the tagged balance, including the one that is entering service, L_2. Here, L_2 is the sum of the expected number of the L_1 requests which were not completed during the first cycle, plus the expected number of new arrivals during the first cycle while the tagged unit was in service.

Thus,

$$L_2 = L_1 e^{-\sigma Q} + \alpha_1 [L_1(S_1+V) + (Q+V)] \tag{21}$$

where α_1 is the average arrival rate from the beginning of the first cycle through the beginning of the second cycle. The actual arrival rate depends on the number of consoles thinking, and hence changes with each new arrival and each service completion. We estimate its mean by averaging its value at the beginning of cycle 1 with its value at the beginning of cycle 2. For simplicity, we ignore the possible unit initially in service.

$$\alpha_1 = \alpha(N-1-\frac{L_1+L_2}{2}) \tag{22}$$

Solving equations (21) and (22) for L_2(and Y_2), and replacing the subscript 2 by i, gives

$$L_i = \frac{L_{i-1}e^{-\sigma Q} + \alpha(N-1-L_{i-1}/2)\,(Y_{i-1}+Q+V)}{1+\alpha(Y_{i-1}+Q+V)/2} \tag{23}$$

$$Y_i = L_i(S_1+V) \qquad i = 2,3,\ldots \tag{24}$$

For a still simpler approximation to Y_i, we can assume that all cycles after the first have the same average length, and set this length equal to

$$Y_B = L_B(S_1+V) \tag{25}$$

The approximations (24) and (25) for Y_i were both used to cost round-robin strategies with parameter values from the M.I.T. operation, and yielded costs within zero to five percent of each other. The approximations compare well with results from simulations and statistics from actual running experience.

COSTING THE MODEL

Analytical studies of time sharing have tended to conclude with derivations of operating efficiency, average number waiting, and average delay[9,11,14]. This leaves the matter of optimum quantum size still ambiguous. We shall approach the problem directly by applying the earlier discussion of cost curves to the model just proposed.

We postulate that a request of size R waiting a time W adds $C_R W$ to the cost accumulated by the operation. The cost rate C_R is chosen to be constant with respect to W, corresponding to the linear case depicted by Figure 1a. This choice permits us to use the preceding expected value arguments.

To express preference for short requests, we let C_R be a monotonically non-increasing function of R. If a request of size R contains k segments for a given Q, then its expected wait, exclusive of its own service, but including its overhead, is

$$W_k = \sum_{i=1}^{k} Y_i + kV \tag{26}$$

A measure of total cost is therefore

$$C = \sum_{k=1}^{\infty} \left[\sum_{i=1}^{k} Y_i + kV \right] \int_{(k-1)Q}^{kQ} C_R f(R) dR \tag{27}$$

In particular, if

$$C_R = \left(\frac{\sigma + \gamma}{\sigma}\right) e^{-\gamma R} \tag{28}$$

and we use approximation (25) for Y_i ($i \geq 2$), then

$$C = Y_1 + \left(\frac{L_B(S_1+V)}{e^{(\sigma+\gamma)Q} - 1} \right) + \left(\frac{V}{1-e^{-(\sigma+\gamma)Q}} \right) \tag{29}$$

The exponential cost function (28) has the effect of steepening the pitch of f(R') while maintaining its exponential form. It is as though we pretend that small requests are more numerous than they really are, then use average wait to measure the performance of the operation.

Expression (29) has been used to compare the cost implications of different choices of Q. Notice that an infinite quantum, corresponding to a FCFS discipline, produces $C=Y_1+V$, independent of γ. In the calculation of C as a function of Q, values for α and σ observed by Scherr[14] were combined with a range of values for N, V, and γ. The results are shown as log-log plots in Figure 4.

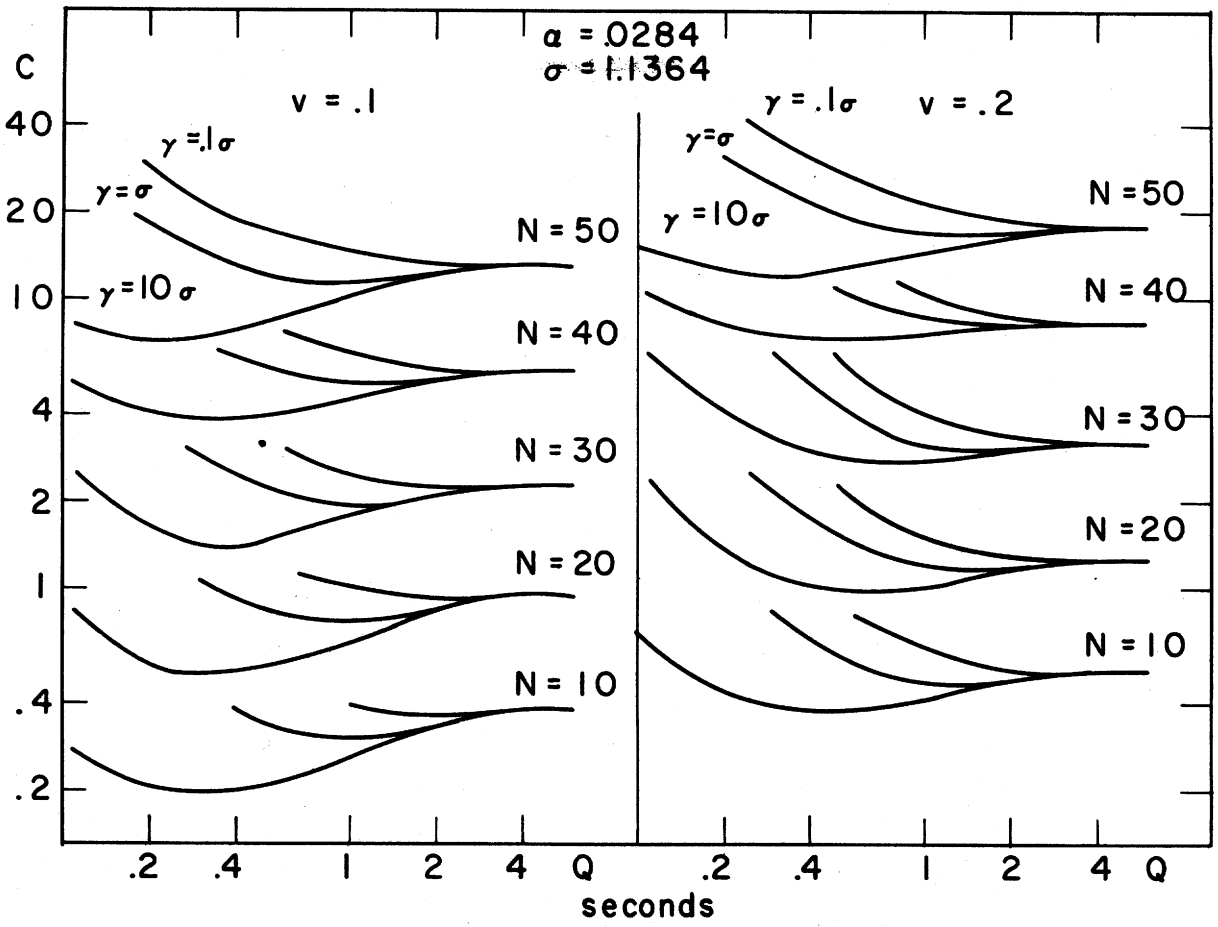


Figure 4. Cost Performances as a Function of Quantum Size; (N and γ parametric)

The larger γ, the more we are advised to favor short requests and the smaller is the optimal quantum. The steepness of the curves in Figure 4 to the left of their minima is due to V, and declines as V approaches zero. This is shown better by the semi-log plot in Figure 5 for $\gamma=\sigma$ and N=30. The relative flatness of the curves to the right of their minima suggests that it is better to be high in the selection of Q, rather than low, except when V is negligible.

As V increases, the maximum cost saving possible from partitioning diminishes, and the optimal quantum grows in size. This is hardly surprising, since overhead is pure cost to the operation. In the absence of overhead (V=0), partitioning is able to favor the short requests without degrading average service. In fact, when V=0, the cost can be reduced by as much as

$$1 - C(Q=0)/C(Q=\infty) = 1 - (1+\sigma/\gamma)^{-1} \qquad (30)$$

a cost saving of 50 percent for $\gamma=\sigma$, up to a theoretical limit of 100 percent as γ goes to infinity. When $V > 0$, however, even though partitioning still benefits some requests, average service must suffer; the smaller Q, the more it suffers.

The requests that benefit are those whose expected wait is smaller than the average FCFS system wait. That is, those requests for which W_k from equation (26) is less than

$$W_{FCFS} = L_B/\sigma' \qquad (31)$$

where L_B is given by equation (17), with $Q=\infty$ and $\sigma'=(1/\sigma+V)^{-1}$. Equation (31), with σ' calculated from equation (14), can be used to infer the average RRS system wait for finite Q. This wait increases monotonically with decreasing Q when $V > 0$.

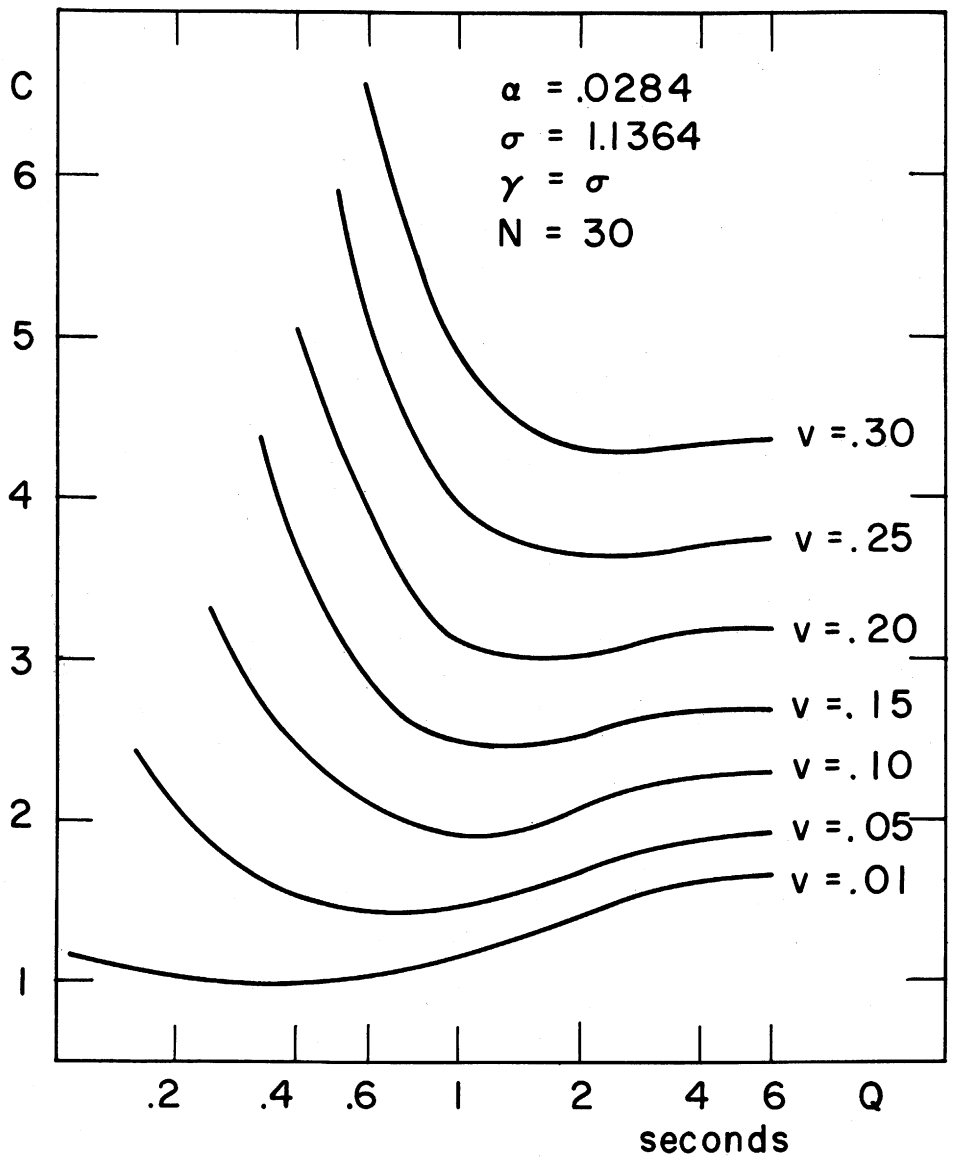


Figure 5. Cost Performances as a Function of Quantum Size; (N and γ fixed)

MULTIPLE LEVELS AND VARIABLE QUANTA

Round-robin scheduling, as we have been discussing it, is a little like traffic flow on a single-lane highway. A faster car must wait until it is safe to pull out and pass a slower car. Throughout the duration of its travel, a slower car delays all of the faster cars that come upon it.

This is in contrast to a multiple-lane highway where the slower cars can stay to the right, out of the way of the speedier ones. The multiple-lane highway has a counterpart in round-robin scheduling. It is a scheduler with more than one level of priority. The multi-level type of scheduler was proposed by Corbató[3] in one of the first papers on time-sharing.

With a multi-level scheduler, if a request is not completed within its quantum, its balance falls to the next priority level and is not served until everything ahead of it has been served, including new arrivals at the higher levels. A request arriving at a higher level (than the one occupied by the request in service) may preempt this request immediately, or at the end of some prescribed time, such as the quantum of the higher level. Each level may have a different quantum associated with it. To reduce overhead, Corbató suggested quantum sizes increasing exponentially for each lower level of priority. He also recommended discriminating against a new arrival whose program size was large, since larger programs have greater overhead requirements than smaller programs. Small programs may be entered at the highest priority level, medium-sized programs at the next higher level, and so on.

Going to multiple levels of priority, and then to different quanta for each level, presents two additional degrees of freedom to the scheduling problem. A third element of flexibility is obtained by allowing the size of quanta to vary with the state of the system. Thus, when there are relatively few consoles waiting, we might lengthen the quantum in order to reduce overhead and response time for the request in service. The request at the end of the line is still served without undue wait.

Conversely, if the system is heavily congested, we might wish to shorten the quantum to allow for the possibility that some of the later requests are small ones. This maneuver does preserve reasonable response times for a priveleged few, but has the unfortunate effect of further degrading the processor just when its speed is needed most.

A quantum that is state-dependent is familiar to those of us who think of our work schedules as round-robin in spirit. We spend as much time as we can on the task of greatest immediate importance, finishing it if possible

without getting too far behind in other responsibilities that accumulate. We make our priority decisions heuristically, and adapt to changing work loads. This flexibility can be imitated by a computer. The decision procedure of the computer need not be rigid, although rigid procedures do have certain advantages. They tend to make analysis simpler, and are generally more economical in computational requirements. Excessive flexibility can cost more than it saves.

CONCLUDING REMARKS

It is time to look back over the road we traveled and raise certain questions that have been put aside. We began by introducing a general method of costing a service operation, and sketched its application to the priority scheduling of a job shop. Then we discussed the simple round-robin scheduling of a time-shared computer system, and used expected-value arguments and linear costing to measure its performance as a function of quantum size. Now we have just mentioned possible extensions to the simple round-robin procedure. Our main energy has been devoted to finding better ways of favoring the short request.

As the lot of the short request improves, that of the longer request must worsen, assuming that average performance gets no better. Does linear costing give adequate attention to the growing wait suffered by the longer request? No, if we believe that the second minute of waiting is worse than the first. To take account of such nonlinearities, we must do nonlinear costing, using curves like (f) of Figure 1.

Nonlinear costing cannot be accomplished analytically with expected-value arguments, but it is easily applied to the results of a simulation or to actual operating statistics. Each wait is costed as it is recorded, by means of an appropriate cost curve. Choice of a cost curve may have to be somewhat arbitrary, but it can also be reasonable. This is illustrated by our selection in the expected-value analysis of a constant cost rate whose logarithm was negatively proportional to size of request.

One approach to nonlinear costing is to postulate for each request a desirable response time or deadline as in the job shop. This leads to cost curves that are step functions. Deadlines are assigned on the basis of name of command, importance of problem, nature of use, or some combination of such factors. Priorities may then be awarded by a c/t rule, where c reflects the probability that a request will miss its deadline, and t is an estimate of the expected time to complete the request. Round-robin scheduling provides a hedge on this estimate. Quantum sizes can vary with the number of users waiting and the imminence of a deadline.

Is shortness of request really a valid criterion for awarding priority? The answer is yes, if we believe that the main purpose of time sharing is to create quick access and brief response times for small users; no, if we prefer to believe that responsiveness should be tailored to individual need. Who should get better service: the highly interactive researcher who is amplifying his creative powers by requesting lengthy statistical regressions and

complex data transformations in rapid succession as though they were simple additions; or the casual user who requests minor editoral changes in his program once every fifteen minutes or so? And what of the user who is doing a little of both?

These are difficult questions. There are at least three ways to answer:

1. We can shrug our shoulders at the multiplicity of different possibilities, and continue to operate in the manner assumed by the previous analysis, hoping that the strategy of favoring the short request is best on the average.

2. We can try to discriminate between more and less interactive users by shifting our attention from request sizes to think times, conjecturing that the length of time a user pauses (or works) between successive requests indicates the quality of service he requires to keep him creative.

3. We can accede to the special needs of highly interactive users with long requests, but insist that they identify themselves by making known their willingness to pay a premium price per unit of computation.

A pricing system can be based either on real money or budgeted computer allotments in dollar units. If the user is able to change his bid on-line, and if the computer favors the highest bidder, then time-sharing assumes the appearance of a one-sided auction market. As in the two-sided stock exchange, the buyer submits either a limit bid for service at a particular price, or a market bid for service at the current price. Like a specialist on the floor of the exchange, the computer can keep the price stable by taking a position; namely, working on its reserve of deferred jobs whenever the price threatens to fall too low or move too abruptly.

If requests are still partitioned under a pricing system, the price paid can influence both a user's quantum and his priority level. Priorities and prices are related concepts. They each serve to allocate limited resources. A pricing-priority system makes the relationship explicit. In the process, it permits the user to be party to the priority decision.

REFERENCES

1. Carroll, D. C., Heuristic Sequencing of Single and Multiple Component Jobs, Unpublished Doctoral Dissertation, Alfred P. Sloan School of Management, M.I.T., June, 1965

2. Cobham, A., "Priority Assignment in Waiting Line Problems," Operations Research, Vol. 2, 1954

3. Corbató, F. J., et. al., "An Experimental Time-Sharing System," Proceedings of the SJCC, 1962

4. Cox, D. R., and Smith, W. L., Queues, Methuen and Wiley, 1961, pp. 76-109

5. Fano, R. M., "The MAC System: A Progress Report," IEEE Spectrum, January 1965

6. Feller, W., An Introduction to Probability Theory and Its Applications, Vol. I, Second Edition, Wiley, 1957, pp. 416-420

7. Greenberger, M., "A Mathematical Study of Priority Assignment" (unclassified), Report of OEG Panel on Problems in Naval Command, Control, and Communications (classified), Office of Chief of Naval Operations, Department of the Navy, 1960

8. Greenberger, M., "The Two Sides of Time Sharing," Datamation, November 1965

9. Krishnamoorthi, B., and Wood, R. C., Time-Shared Computer Operations with Both Interarrival and Service Times Exponential, System Development Corporation, SP-1848/000/00, 30 October 1964

10. McCarthy, J., "Time-Sharing Computer Systems," Computers and the World of the Future (M. Greenberger, Ed.), M.I.T. Press, 1964

11. Patel, N., A Mathematical Analysis of Computer Time-Sharing Systems, Master's Thesis, Alfred P. Sloan School of Management, M.I.T., June 1964

12. Rothkopf, M., "Scheduling Independent Tasks on One or More Processors," Interim Technical Report No. 2, Operations Research Center, M.I.T., January 1964

13. Saltzer, J. H., CTSS Technical Notes, Project MAC-TR-16, M.I.T.

14. Scherr, A. L., An Analysis of Time-Shared Computer Systems, (Doctoral Dissertation, Department of Electrical Engineering), Project MAC-TR-18, M.I.T., June 1965

15. Takacs, L., Introduction to the Theory of Queues, Oxford, 1962, Chapters 4 and 5

CS-TR Scanning Project
Document Control Form

Date : 12/11/95

Report # LCS-TR-22

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 36 (43 -images)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☒ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

IMAGE MAP! (1-36) UNNUMBERED ADDENDUM, BLANK, TITLE, AND BLANK PAGES, i, UNNUMBERED, iii-iv, 1-28 (37-43) SCANCONTROL, COVER, FUNDING AGENT, DOD, TRGT'S (3)

Scanning Agent Signoff:

Date Received: 12/11/95 Date Scanned: 1/18/96 Date Returned: 1/19/96

Scanning Agent Signature: Michael W. Cook

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Massachusetts Institute of Technology Project MAC	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

The Priority Problem

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Technical description of computer scheduling

5. AUTHOR(S) *(Last name, first name, initial)*

Greenberger, Martin

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
November 1965	34	15

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
Office of Naval Research, Nonr-4102(01) b. PROJECT NO. Nr-048-189 c. d.	MAC-TR-22 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES

Qualified requesters may obtain copies of this report from DDC.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301

13. ABSTRACT

Priority decisions arise whenever limited facilities must be apportioned among competitive demands for service. A priority operation of contemporary interest is scheduling a time-shared computer among its concurrent users. Service requirements are not known in advance of execution. To keep response times short for small requests, service intervals are partitioned and segments are served separately in round-robin fashion. A mathematical analysis pinpoints the tradeoff between overhead and discrimination, implicit in this procedure, and allows alternate strategies to be costed. Extensions of the simple round-robin procedure are suggested, the objectives of time sharing are reviewed, and implications are drawn for the design of future priority and pricing systems.

14. KEY WORDS

Computer
Machine-aided cognition
Multiple-access computers
On-line computer systems
Real-time computer systems
Scheduling
Time-sharing
Time-shared computer systems

DD FORM 1 JAN 64 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 1/18/1996

M.I.T. Libraries
Document Services