Properties of delay-cost scheduling in time-sharing systems

by P. A. Franaszek
R. D. Nelson

We consider properties of time-sharing schedulers with operations based on an economic measure termed the *delay cost*, and relate these to scheduling policies such as those used in VM and MVS. One of these policies, deadline scheduling, is shown to be potentially unstable. We develop delay-cost schedulers that meet similar performance objectives under quasi-equilibrium conditions but which are stable under rapidly varying loads.

1. Introduction

The purpose of a scheduler in a time-sharing system is to allocate resources to jobs in order to conform to priority objectives while maintaining efficient system operation. These dual goals sometimes result in conditions that require exceptional handling. For example, in some systems, jobs are scheduled according to a priority scheme, provided that certain system resources are not overutilized. To maintain efficient operation, however, might require scheduling a lower-priority job before one of higher priority if the memory requirements of the high-priority job are so large as to cause thrashing. Issues associated with efficient operation often depend on the specific system architecture, such as the structures of main memory and of I/O subsystems. Additionally, the way priorities are assigned to jobs also varies from system to system. Generally, however, scheduling algorithms for time-sharing systems are intended to be easily implementable and to have features that provide flexible system control.

We consider here a scheduling paradigm, initially proposed by Greenberger [1], called *delay cost*. This approach defines an economic measure by which the system is charged for holding jobs. Associated with each job class is a *delay-cost function* of time, which determines the cost charged to the system on the basis of the total time each job of that class has been held in the system. The aim of the scheduler is to minimize the total cost charged. A good heuristic for minimizing the cost (which we rederive below) is simply to schedule that job whose ratio of marginal delay cost to required system resources is highest. We call this heuristic the *delay-cost-ratio* algorithm. An analogy can be drawn here between a set of jobs to be processed by the computer system and a set of loans that must be repaid. The job scheduled in this analogy corresponds to the loan that has the highest interest rate.

Throughout the paper, we make the simplifying assumption that once execution of a job has started on a given processor, it is serviced on that processor until completion. In some systems, a scheduled job is allocated

0018-8646/95/$3.00 © 1995 IBM

some number of cycles on a processor and may go through multiple rounds of scheduling. For example, in the IBM VM/HPO operating system (virtual machine with high-performance option), a job is initially considered to be in class 1. If it does not complete execution with the resources granted to jobs in this class, it is reclassified and scheduled as a job in class 2. Subsequently, the job may go through multiple scheduling phases. For such a system, the sequence of tasks presented to the scheduler may not correspond to the sequence of submitted jobs, since jobs may change class during execution or be scheduled more than once. In our model, we assume that when a job enters a new round of scheduling and execution, it is as a new job of possibly different class. That is, we are considering the scheduling of the work units or tasks that are normally seen by a scheduler in an operating system.

The delay-cost paradigm provides a convenient framework for specifying a scheduler objective function, which one can then attempt to minimize via appropriate scheduling algorithms. Most schedulers used in practice have no such explicit objective, using instead such notions as deadlines or service rates, which are implicit in the scheduling algorithms. A disadvantage of this latter approach is the difficulty in extending it to systems with characteristics other than those originally envisioned. We show here that appropriate delay-cost functions and optimization algorithms can yield, under steady-state conditions, performance analogous to that obtained from techniques based on deadlines or rates of service, while avoiding potential pitfalls such as instabilities under varying loads.

We should mention that a scheduler for a large time-sharing system is a complex program with features not considered here, including subroutines based on estimates of working sets and I/O requirements for executing tasks. The results here do not cover the overall design but instead concentrate on an important aspect, namely the algorithms used to determine the order of execution as a function of system load, and properties of the delay-cost measure that make it suitable as a basis of design.

The following is a synopsis of the paper. Section 2 considers the issue of a scheduler objective function, and derives the delay-cost form. Section 3 considers the problem of formulating scheduling algorithms to minimize the total delay cost. A variety of approaches are described, the first of which is similar to that obtained by Greenberger, corresponding to the loan-repayment analogy mentioned above. Other approaches considered include two that correspond to those used in the IBM MVS and VM operating systems. We describe how schedulers in these systems may be regarded as heuristics for minimizing some implied delay cost for tasks being scheduled. In Section 4, we concentrate on the form of deadline scheduling utilized in the VM/HPO operating system. We show that the deadline form is potentially unstable under rapidly varying loads. This problem can be avoided by determining the delay-cost function implicit in the scheduler, then using a different scheduling heuristic to minimize this cost. We derive a set of delay-cost curves that correspond to the implied goal of the scheduler and then validate the results through simulation. We are interested here in what happens under various dynamic conditions, and we investigate the behavior of the deadline algorithm and its corresponding delay-cost analogue for a particular sequence of jobs. The results indicate that the two forms have essentially equivalent performance under steady-state conditions, but the delay-cost form avoids instabilities under rapidly varying loads. In Section 5, we state our conclusions, and the Appendix contains derivations of various properties of optimal delay-cost schedulers.

2. Schedulers and objective functions

This section considers some scheduler properties and discusses a framework for scheduling based on what we term the *delay cost*. This framework yields heuristics that operate by assigning a delay-dependent priority to the jobs in the system and scheduling the highest-priority job. The difference between this and the usual approach to assigning priorities is that the priorities here are quantities derived from a cost formula.

- *Preliminary assumptions*

The workloads assumed in this paper are broadly characterized by the property that the times of job arrivals and their work requirements can be specified only probabilistically. As mentioned above, each job presented to the system may be scheduled more than once as it consumes processing time and other resources. Certain of the analytical results in this paper are obtained under the assumption that it is adequate to characterize the rounds of scheduling as being statistically independent; that is, that the load can be approximated by a sequence of jobs with independent arrivals, each of which leaves the system after a single round of execution. We also assume that jobs assigned to a processor complete their processing in the order in which they are scheduled. In an actual system, a job's execution may be interrupted by events such as page faults or time-slice expiration. We ignore this, as our model jobs are assumed to run to completion without preemption. Under moderate to heavy load conditions, where the satisfactory operation of the scheduler is most critical, the time a job spends in service is small with respect to the total time it spends in the system, so that interruptions such as page faults do not greatly affect overall response times.

We further restrict our attention to scheduling policies having the following properties:

1. Scheduling decisions are causal, i.e., based upon only the current state of the system and knowledge of the past.
2. The policy is work-conserving; i.e., no processor is idle if there are jobs waiting for service.
3. The policy is time-stationary and deterministic; i.e., any time the system is in the same state with an identical set of jobs to be scheduled, the scheduler makes the same job selections.

Scheduling policies found in current computer systems such as VM [2] and MVS[1] generally satisfy these properties. An in-depth treatment of such scheduling policies for Poisson arrival streams can be found in [3], and issues concerning the synthesis of schedulers to meet certain objectives are discussed in [4] and [5].

• *Delay-cost scheduling*

One way of stating an objective for the scheduler is to define a function, termed the *objective function*, which maps scheduling policies to real numbers. We say that one policy is better than another if it achieves a lower value for this function for the same sequence of jobs. As is shown below, it is not possible in general to construct a policy that is *optimal*, i.e., which minimizes the value of this function for all job sequences. Thus, it is usually necessary to employ a heuristic. Each such heuristic may operate as though enforcing a different set of time-dependent priorities, sometimes with substantially different results (e.g., lack of stability). This suggests that an objective function is desirable for reasons of both generality and flexibility. In light of the desirability of having such functions, it is surprising to find that their specification is missing from the definition of many schedulers (for additional discussion of this point, see the conclusion in [6] and Chapter 3.7 of [7]).

We now provide some definitions. Let $J = \{1, 2, \cdots, N\}$ be the sequence of jobs to be processed, where job j arrives at time t_j, and $i > j$ implies that $t_i > t_j$. Let t'_j be the time at which job j is scheduled, and let e_j be the time it leaves. The set of times $\{t'_j\}_k$ for a set of jobs J_k is then a *schedule* for this set. Let $x_j = e_j - t'_j$ be the *service time* of job j. We denote its expectation by $\bar{x}_j$ and assume that this is dependent only on the job's class. At time t, for $t > t_j$, the amount of time spent in the system by job j, denoted $T_j(t)$, is $\min[(t - t_j), (e_j - t_j)]$. The *response time* for job j is given by $T_j = e_j - t_j$. We denote the scheduler objective function by $F(J, t)$, a real-valued function of time t. The *waiting time* for a job is $e_j - x_j - t_j$.

We restrict our attention to functions $F(J, t)$ that are dependent only on the response times for jobs in J which enter the system before time t. We refer to $F(J, t)$ as the *cost* incurred by a schedule at time t. We further require the following:

1. The function $F(J_i, t)$ is defined for any subset J_i of J [so that, for example, $F(J, t)$ can be used to evaluate the effect of a schedule on any class of jobs].
2. If $J_1, J_2, \cdots, J_n$ are disjoint subsets of J, where $\cup_{i=1}^{n} J_i = J$, then $F(J, t)$ can be determined from the set $\{F(J_i, t)\}$, $i = 1, 2, \cdots, n$.

The latter requirement ensures that the overall cost of a schedule for a group of users can be determined from their individual costs. We assume

$$F(J, t) = F(J_1, t) \; \zeta \; F(J_2, t) \; \zeta \cdots \zeta \; F(J_n, t), \tag{1}$$

where ζ is a commutative and distributive operator, either addition or multiplication. These two operators are equivalent via the use of logarithms, and here we take this operator to be addition. Without loss of generality, we assume $F(j, t) \geq 0$, where $F(j, t)$ is the cost of an individual job. Since jobs that have not arrived by time t have no effect on the objective function, and jobs have no further effect once finished, $F(j, t) = 0$ for $t \leq t_j$, and $F(j, t) = F(j, v)$ for t, v such that $e_j \leq t \leq v$. We also assume that $F(j, t)$, for $t_j < t < e_j$, is differentiable in t, and generally require that the incremental amount added to the objective function for any job j increase with the amount of time this job spends in the system. This implies that $\partial F(j, t)/\partial t > 0$, for $t_j < t < e_j$ (we use > 0 rather than ≥ 0 to ensure first-come/first-served in class). The assumption that $F(j, t)$ is differentiable allows us to represent it as an integral of some function; moreover, it is clear from the above properties that the value of $F(j, t)$ changes only while job j is in the system. Thus, we can write $F(j, t)$ as

$$F(j, t) = \int_0^{T_j(t)} C(y)\, dy, \tag{2}$$

where $C(\cdot)$ is a nonnegative function termed the *delay-cost function* [1]. We note here that $C(y)\, dy$ can be interpreted as the incremental cost charged to the system for holding job j longer than y units of time (seconds). Using Equation (1) allows us to write the total cost charged to the system for processing jobs in set J as

$$F(J) = \sum_{j \in J} \int_0^{T_j} C(y)\, dy, \tag{3}$$

where $F(J) = \lim_{t \to \infty} F(J, t)$. We can rewrite Equation (3) to account for different job classes by extending our definitions. Suppose there are K distinct job classes and that the set of jobs from J that belong to class k is given

[1] *OS/VS MVS Resource Measurement Facility (RMF) Reference and User's Guide*, IBM Data Processing Division, White Plains, NY (no longer in print).

by J_k. Let $t_{j,k}$, $e_{j,k}$, and $T_{j,k}(t)$ be the arrival, exit, and total amount of time spent in the system at time t for the jth job of the kth class. The response time for the jth job of the kth class is given by $T_{j,k} = \lim_{t\to\infty} T_{j,k}(t)$. Equation (3) is then written as

$$F(J) = \sum_{k=1}^{K} \sum_{j\in J_k} \int_0^{T_{j,k}} C_k(y)\, dy, \tag{4}$$

where $C_k(\cdot)$ is the delay-cost function for the kth class.

The form of Equation (4) has been suggested or used previously (see for example [1, 8, 9]). The purpose of the above derivation is to expose the assumptions on which it is based. As discussed below, these assumptions do not always hold in practice.

A primary assumption used in the derivation of Equation (4) is that $F(\cdot)$ can be applied to any subsequence in the job stream, hence that jobs are essentially independent. Examples of cases in which this is not true include sets of jobs that must be scheduled together because the overall cost is related to the time required for the full set to complete, and systems with constraints on the percentage of work to be granted to a given subclass of jobs. We do not discuss this issue further here, but merely mention that the delay-cost measure can often be adapted to obtain reasonably good schedules. An example is discussed in the subsection on rate-of-work schedulers.

The goal of the system is to minimize the value $F(J)$ given by Equation (4), i.e., to minimize the total cost charged to the system for processing the jobs in J. This minimization is performed over all possible ways to schedule jobs in J, subject to the constraint that no job is scheduled before its arrival. If the arrival times and service times were known in advance, one could hypothetically minimize $F(J)$ by evaluating all possible ways of scheduling jobs and selecting one with a minimal cost. In a real system, of course, the arrival and service times of jobs are not typically known in advance and may not even be known probabilistically.

We denote the value of the objective function when policy S is applied to the jobs in J as $F^S(J, t)$. It is clear that the choice of a scheduling policy influences the finishing times of the jobs; thus, in general, $F^S(J, t) \neq F^{S'}(J, t)$ for two different scheduling policies S and S'. To enable a selection among different scheduling policies, *by definition* of F, we say that policy S is better than policy S' at time t if $F^S(J, t) < F^{S'}(J, t)$. An optimal policy for objective function F at time t is a policy S^* that satisfies $F^{S^*}(J, t) = \min_S F^S(J, t)$. An optimal policy for objective function F is an optimal policy at time t in the limit as $t \to \infty$, providing such a limit exists.

Suppose job arrivals and service demands are not known in advance but are modeled by random variables, where jobs of the same class have service demands given by a class-dependent distribution that does not vary with time. The value of the objective function for this case is given by

$$F^S(J) = E\left[\sum_{k=1}^{K} \sum_{j\in J_k} \int_0^{T^S_{j,k}} C_k(y)\, dy\right], \tag{5}$$

where $T^S_{j,k} = e^S_{j,k} - t_{j,k}$ is the time spent in system for the jth job of the kth class when scheduling policy S is used.

3. Heuristics

This section discusses some heuristic approaches to minimizing the delay-cost measure $F(J)$. These include techniques loosely modeled on algorithms encountered in MVS[1] and VM [2]. When considering these, one should understand that the analysis is aimed at obtaining an improved understanding of a *class* of algorithms rather than of any specific scheduler implemented in an operating system.

We begin by considering examples of delay-cost functions and how these affect scheduling decisions. Techniques for delay-cost reduction are then examined. These include a) the delay-cost-ratio (DCR) algorithm, b) rate-of-work schedulers, related to the approach used in MVS, and c) deadline scheduling, a variant of which appears in some VM systems.

• *Delay-cost functions*

We now consider some specific examples of delay-cost functions $C_k(\cdot)$.

1. *Class-dependent constant* Here the delay-cost function is a class-dependent constant; thus, the cost charged to the system is directly proportional to the response times of jobs. This form was used by Klimov [8, 9] and further studied in [10]. Although the above cost function is convenient for analysis, it has the drawback that simple cost-minimization algorithms may not yield generally desirable properties such as first-come/first-served (FCFS) within class. A consequence is that individual jobs may never receive any service.
2. *Class-dependent linear* Here the cost charged increases as the square of the response times. It is shown below that algorithms employed in some deadline schedulers, such as the one found in VM, can be regarded as heuristics that minimize delay costs of this form. If the DCR scheduling algorithm (described in the following subsection) is used, the delay-cost ratios $C_k(\cdot)/\bar{x}_k$ are then linear functions. An analysis of the mean response times for linear delay-cost functions when $F(j, t_j) = 0$ was derived in [11] and further considered in [12]. Bounds on the mean response time

for nonzero intercepts were derived as special cases in [13], and an exact, closed-form expression for heavy traffic is given in [14]. We use the latter results in Section 4 while discussing deadline scheduling in further detail.

3. *Arbitrary increasing function* A delay-cost function of this form may be viewed as producing the criterion used in some rate-of-work schedulers (e.g., MVS, as shown below). A nonlinear, bounded cost function can be used to ensure properties such as the exclusion of a job class once the system load reaches a given level. Bounds on the mean response time for related priority schemes (e.g., priorities given by concave functions) can be found in [13] and [15].

In the following, we restrict our attention to delay-cost functions that are increasing with time and unbounded. This implies that it becomes incrementally more expensive for the system to keep a job the longer it stays, which guarantees that, if the system is not overutilized, every job is eventually processed. Jobs of the same class are assumed to have similar service times, so it is not surprising that an optimal delay-cost scheduler schedules jobs within a class in an FCFS manner. Intuitively, this follows from the fact that an optimal algorithm would always select the most expensive job to receive processing first. This property (Proposition 1) of optimal delay-cost schedulers is proved in the Appendix.

• *A delay-cost heuristic*

Let y_k be the time spent in the system for the oldest class-k job. From the FCFS property, we know that only the oldest job from each class must be examined in making scheduling decisions. Suppose that one of the next two jobs to be scheduled is from class k and the other from class k', and attempt to determine which class to select first. Let y_k and $y_{k'}$ be respectively the residence times for the two jobs that are the oldest (i.e., have the longest residence times) in the two classes. Then, the class-k job should be scheduled first if

$$E_{(x_k,x_{k'})}\left[\int_0^{x_k} C_k(y_k+z)\,dz + \int_0^{x_k+x_{k'}} C_{k'}(y_{k'}+z)\,dz\right] < E_{(x_k,x_{k'})}\left[\int_0^{x_{k'}} C_{k'}(y_{k'}+z)\,dz + \int_0^{x_{k'}+x_k} C_k(y_k+z)\,dz\right], \tag{6}$$

where $E_{(x_k,x_{k'})}$ denotes the expectation. If $\bar{x}_k$ and $\bar{x}_{k'}$ are sufficiently small with respect to y_k and $y_{k'}$, and if $C(\cdot)$ varies sufficiently slowly, Equation (6) implies that the oldest member of class k should be scheduled ahead of that in k' if $C_k(y_k)/\bar{x}_k > C_{k'}(y_{k'})/\bar{x}_{k'}$ (see [1] for a different derivation; this has also been termed the μC *rule* [7]). In the Appendix, we provide a formal proof for this property (Proposition 2), which we call *pairwise optimality*. This suggests the following heuristic: Schedule the oldest job of class k if it has the maximal value of $C_k(y_k)/\bar{x}_k$. Note that this policy satisfies FCFS within class. We call this the delay-cost-ratio (DCR) algorithm, and the value of $C_k(y_k)/\bar{x}_k$ is called the DCR value of the oldest class-k job. In the analogy of loan repayment and delay cost, $C_k(y_k)$ corresponds to the amount of money charged per unit of time during which the loan is outstanding. We assume that this amount increases with time. The value of $\bar{x}_k$ corresponds to the amount of the loan owed, and $C_k(y_k)/\bar{x}_k$ to the current interest rate. The DCR algorithm suggests simply repaying the loan with the highest interest rate (see [16] for a discussion of bounds for the delay cost). We note that the DCR algorithm depends upon only the first moment of service time for the different classes of jobs. This is a result of a heavy-traffic limit that is used in establishing Proposition 2. For lighter loads, however, higher moments of the class service-time distribution could be included in the heuristic, as is shown in Equation (A1).

Proposition 3 in the Appendix shows that under heavy traffic, Poisson arrivals, and linear delay-cost functions, the DCR algorithm satisfies $C_k(\bar{T}_k)/\bar{x}_k = \beta$, where $\bar{T}_k$ is the mean response time for class k, and β is a constant that measures the load of the system. One expects this relationship to hold approximately under less restrictive conditions, since the DCR algorithm tends to minimize the maximal value of $C_k(y_k)/\bar{x}_k$ and thus tends to establish a common value for $C_k(\bar{T}_k)/\bar{x}_k$. The value of β plays a crucial role in delay-cost scheduling and is the main mechanism we use in establishing a mapping between delay-cost scheduling and other policies, such as deadline scheduling.

Before ending this subsection, we make a few comments about the delay-cost heuristic. The first is that differentiation between job classes manifests itself in two ways. One is in the specification of the delay-cost functions by the system implementers. The other is on the basis of the average service requirements of the different classes; here, as the heuristic implies, large jobs are penalized in relationship to small jobs. This follows from the fact that in the attempt to minimize the total delay cost of the jobs in the system, it would often be the case that several shorter jobs could be completely serviced during the time needed to finish the execution of a larger job. One can interpret the DCR value as being a cost rate, i.e., cost per unit work. Thus, the DCR algorithm schedules the job having the highest cost rate. An alternative interpretation of the DCR value is that of a *price* for processor cycles. A job willing to pay a higher price is thus more desirable for execution.

In terms of implementation, the heuristic suggests that one keep a queue for each class of jobs. Scheduling decisions can then be made by calculating the DCR for the

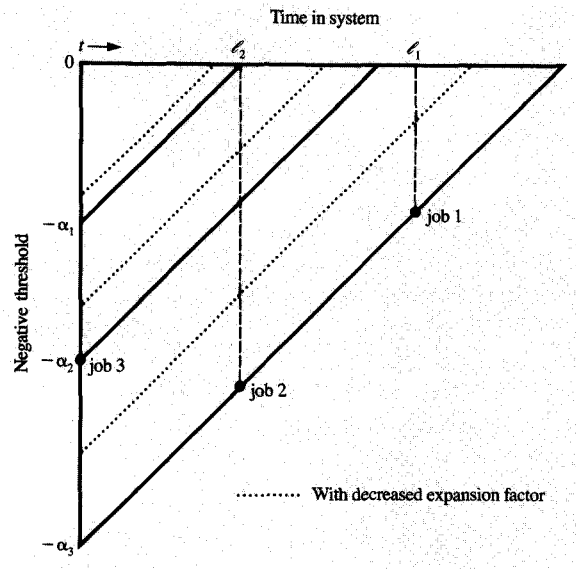

Figure 1

Graphical representation of a deadline scheduler.

job at the head of each queue and selecting the one with the maximal value. Jobs not at the head need not be inspected. As a final remark, we note that the value of β defined above is a measure of the load of the system. The load seen by a class-k job that has been in the system for y_k seconds is given by $C_k(y_k)/\bar{x}_k = \beta_k$. Decisions made by DCR schedulers then can be viewed as scheduling that job corresponding to the highest perceived load. This observation is used in the following two subsections when establishing a correspondence between DCR schedulers and rate-of-work and deadline schedulers.

- *Rate-of-work schedulers*

Some schedulers (the one in MVS is an example) operate by allocating processor time to members of each job class at a rate that is a function of system load. The schedulers determine which job should receive the next time slice, and a job that fails to complete execution during its time slice must be scheduled again. More precisely, the schedulers attempt to give processing to all members of job class k at a rate $R_k(z)$, for $k = 1, 2, \cdots, K$, where z is a parameter that represents the system load. These functions $R_k(z)$ are strictly decreasing with z over the interval $0 \le z \le z_k < \infty$ and identically equal to 0 for $z > z_k$, where the z_k are class-dependent values. Thus, the value of z_k represents the load past which a given job class is denied service. Suppose job i (of class k) is currently receiving processing at a rate of v_i. This corresponds to a load value of $z_i = R_k^{-1}(v_i)$. [Let $R_k^{-1}(v)$, for $v > 0$, denote the inverse of $R_k(\cdot)$, and define $R_k^{-1}(0) = z_k$.] The scheduler then chooses the job that corresponds to the maximal z_i over all jobs in the system. At any time, there is a value z^* that is the load of the system. This is analogous to the value of β previously defined for DCR schedulers. Let m_k be the number of class-k jobs in the system at a given time, and let the total rate of work that can be supplied by the system be given by $\mathbb{P}$. The parameter z^* is then defined to be the value of z that satisfies

$$\sum_{k=1}^{K} \bar{R}_k(z) m_k = \mathbb{P},$$

where $\bar{R}_k(z)$ is the average value of $R_k(z)$ over class k.

The above is similar to the operation of the DCR delay-cost heuristic described above, in that both operate by scheduling the job corresponding to the highest perceived load. There is a difference, however, between practical implementations of the schedulers: In the DCR case, each job is viewed as being scheduled only once, whereas in the rate-of-work scheduler, jobs may be scheduled many times. The two approaches may be brought into closer correspondence by assuming the viewpoint that upon expiration of its time slice, a job reenters as a "new" job. Upon reentering the system or at initiation, the job is assigned a rate of work (for reentering jobs, this could be the rate just after finishing the previous processing slice) which then changes over time. Let $v_i(t)$ be the rate of job i at time t. Roughly speaking, a corresponding DCR scheduler is characterized by the following delay-cost functions:

$$C_k(t) = \bar{x}_k R_k^{-1}[v_i(t)] \qquad k = 1, 2, \cdots, K. \tag{7}$$

- *Deadline schedulers*

The deadline schedulers (DSs) considered here operate by assigning a parameter, termed a *deadline*, to each new job as it enters the system. This parameter, which is generally a function of the job class and system state, is then used to order the jobs waiting for execution. The deadline is sometimes viewed heuristically as the time by which a job should complete its processing. In the absence of exceptional conditions, such as memory thrashing, jobs are scheduled according to their deadlines, which are unchanged once assigned. A DS of this form is included in a current version of VM [2].

A typical function used for the deadline is given by

$$t_j + \mathbb{E}H_k \qquad 1 \le k \le K, \tag{8}$$

where t_j is the time of job arrival; $\mathbb{E}$, termed the *expansion factor*, is a function of the response times of the job classes (there are three classes in some versions of VM); and H_k is a class-dependent constant. A job of class k

entering the system at time t_j is given the deadline specified by (8) and placed in the system queue in increasing deadline order. Jobs in the queue are then selected in that order for execution whenever their working sets fit into memory. In VM, jobs entering the system are initially assigned to Class 1 for some limited amount of processing and are then rescheduled as Class 2 (and eventually as Class 3).

In the following, we consider the class of a job when determining its deadline and assume that H_k increases with k. The function $\mathbb{E}$ is a linear weighting of the current average class response times, which are typically determined by measuring job departures over some interval (this is further discussed in Section 4). Specifically, suppose the current class-k mean response time is $\bar{T}_k$; then, the expansion factor is calculated as follows:

$$\mathbb{E} = \sum_{k=1}^{K} \frac{a_k \bar{T}_k}{\bar{x}_k}, \tag{9}$$

where the a_k are constants that determine the weights given to each class, and $\bar{T}_k/\bar{x}_k$ is the expansion of the class-k response time due to other jobs in the system. That is, if only one job were present, $\bar{T}_k/\bar{x}_k$ would equal one. We call the value $\mathbb{E}H_k$ a *threshold* value, and we denote it by α_k. As mentioned above, in deadline scheduling, a queue of jobs is maintained in increasing deadline order. Thus, a job's deadline determines its relative response time. Roughly speaking, the response time of a job is proportional to its deadline, for a given value of the expansion factor.

We now provide an intuitive argument that deadline scheduling can be viewed as a heuristic for minimizing the total delay cost for a particular form of delay-cost function. To show this, it is convenient to view deadline scheduling in terms of priority scheduling. Once a job is placed in the queue, its position relative to other jobs already in the queue is not changed. Each job approaches its deadline at a constant rate of one second per second elapsed. Differentiation between different classes of jobs with respect to allocating processing arises only from differences in their initial queue placement, which is determined by the initial deadline calculation. A scheduling decision consists of scheduling that job with $\min[\alpha_k - (t - t_i)]$, where k is the class of job i. For the case in which $\alpha_k - (t - t_i) > 0$, this schedules the job closest to its deadline.

Figure 1 shows how this operation might be represented graphically in a system under constant load. The solid lines $\{-[\alpha_k - (t - \ell_i)]\}$ correspond to the same fixed load, which implies the value of $\mathbb{E}$ and values of α_k. In this example, job 3 (a class-2 job) has just entered the system at coordinates $(0, -\alpha_2)$ and will progress upward to the right toward its deadline $(\alpha_2, 0)$ at a rate of one second per second elapsed. Job 1 (a class-3 job) entered the system ℓ_1 seconds ago, and job 2 (another class-3 job) entered ℓ_2 seconds ago. Since $\alpha_2 \leq \alpha_3$, class-2 jobs are initially placed higher than those of class 3. All move up to the right at the same constant rate. The job at the top of the queue (the next job to be scheduled) is the highest one—job 1—followed by job 3 and then job 2.

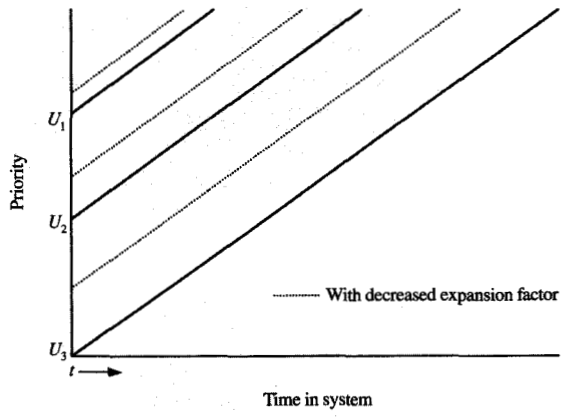

Figure 2

Deadline scheduling viewed as priority scheduling.

In terms of priority scheduling, the above suggests that the priority of a job of any class increases linearly in time with unit slope, and that differences between classes are determined by an initial value. Scheduling decisions, viewed in this manner, consist of scheduling the job with the highest priority.

In **Figure 2**, we show how the above description of a DS might be viewed in terms of priorities. Let the initial priority, or *offset*, for a class-k job be denoted u_k. An arriving class-k job can then be thought of as having an initial priority of u_k, which increases at a unit rate. Scheduling decisions consist of selecting the job having the highest priority value, which, in a deadline scheduler, is the job closest to its deadline. It is convenient to shift the linear priority curves so that the lowest class starts out with a value of zero. Then the relationship between the threshold values of the DS and the offset values for the priority scheduler can be expressed as $u_k = \alpha_K - \alpha_k$ for $1 \leq k \leq K$, where K is the lowest-priority class.

This view of deadline scheduling is not complete, however, in that it ignores the effect of changes in the system load that modify the deadline calculation and thus, equivalently, modify the initial priority values. The effect of a decrease in the value of the expansion factor $\mathbb{E}$ is

shown by the dotted lines in Figures 1 and 2. Here, the decrease in the threshold values α_k for the DS corresponds to an increase in the offsets u_k for the equivalent priority scheduler. The manner by which deadlines change as a function of the load of the system leads to an interesting correspondence between deadline scheduling and DCR scheduling, which we now address. First, however, we note that the form of DS implied by Equation (8) is potentially unstable and may lead to anomalous behavior. We analyze these behavioral characteristics and establish the relationship between DSs and DCR schedulers in greater detail in Section 4.

We now show that if the deadline computation is given by Equation (8), then, for slowly varying loads, scheduling decisions are approximately equivalent to a DCR minimization with linear delay-cost functions. As Equation (8) implies, deadlines calculated by the system change proportionately with $\mathbb{E}$, where the constants of proportionality are given by the values H_k. This implies that, for each class of jobs, the mean waiting time $(e_j - x_j - t_j)$ changes with approximately the same constant of proportionality. A result of Proposition 3 in the Appendix is that under heavy traffic conditions, with Poisson arrivals, $C_k(\bar{T}_k)/\bar{x}_k = \beta$ for all classes, where β is a measure of the load of the system and is thus analogous to $\mathbb{E}$ in the deadline calculation. If we suppose that $C_k(t)$, the delay-cost function for class k, equals $s_k t$ (constants s_k are the factors in the linear delay-cost functions), then as β varies slowly, the mean response times of the classes change proportionately to $\bar{x}_k/s_k$ in a DCR scheduler. Thus, equating $\bar{x}_k/s_k$ and H_k yields a DCR scheduler with mean response times similar to those obtained by the DS. In Section 4, we make these statements mathematically precise.

A principal result from the above observation is that if the deadline computation is of the form of Equation (8), a delay-cost-based scheduler may be constructed whose decisions are similar to those of a DS under slowly varying loads but which, we later show, avoids the instabilities associated with deadline scheduling under rapidly changing loads. Here the formulation of an objective function for the scheduler permits the consideration of schedulers which have similar overall properties, but which are superior under some circumstances.

4. Delay-cost version of a deadline scheduler

This section provides an analysis of the performance and behavior of DSs. The results show that although these schedulers can work well under conditions of slowly varying load, they can exhibit anomalous behavior and instabilities when attempting to adapt to rapidly changing conditions. The following subsection contains a description of how this behavior can occur and formalizes the correspondence between DSs and DCR schedulers that was outlined above. The equations obtained were then used to drive a simulation to compare the performance of these two schedulers. This simulation demonstrated that anomalous behavior and instabilities are, in fact, found in DSs but, notably, are missing from the equivalent DCR scheduler. Finally, this section concludes with modifications to a DS that may achieve similar performance objectives but have the advantage of stability and lack of anomalous behavior.

• *Stability*

A scheduler provides a means of controlling a system in order to obtain favorable characteristics for a variety of loads. As in any control system, stability is a key issue. A scheduler is a nonlinear system, so there are a variety of ways of defining stability (Reference [17] notes at least 28 definitions in use). One way of indicating a lack of stability is to show that there exist a system state and an input that produce undesirable oscillations. We later show by simulation that such oscillations can occur. Another way of demonstrating instability is via the presence of positive feedback in the control policy at equilibrium points, so that perturbations are not self-correcting. We show that there are instances of positive feedback in DSs. First, however, we discuss properties of DSs that lead to such potential instabilities.

As noted above, once a job is placed in the queue by the scheduler, its position relative to other jobs already in the system remains unchanged. Other important properties related to stability are that a) the expansion factor at any given time is a function of the response times for jobs that completed service over a previous time interval that we term the *sampling period*, and b) the value of the expansion factor can undergo rapid change.

We now consider some qualitative aspects of DS behavior. Suppose that there are two classes of jobs, corresponding to edit and batch jobs, with edit jobs having substantially smaller service times than batch jobs and requiring shorter response times. Properties that would be expected to hold in a good scheduler, but which can be violated under rapidly varying loads for a DS include the following:

1. *FCFS within classes.*
   (This fairness criterion might be violated under conditions of decreasing load. For example, if the expansion factor is decreased, an arriving job may be assigned a smaller deadline and thus may be placed ahead of jobs in its class already in the queue.)
2. *Edit jobs have smaller response times than batch jobs.*
   (This might be violated under either increasing or decreasing load conditions. Consider a batch job that has just been placed in the queue and suppose that the expansion factor increases by a large amount. It might

then be the case that a newly arriving edit job, having a larger deadline, is placed after it in the queue. The edit job is scheduled after the batch job and thus has a longer response time. This phenomenon might also occur under decreasing load conditions if the expansion factor undergoes a sufficiently large decrease.)

3. *Under increasing load, edit jobs get a larger proportion of the CPU resources.*
(One way to see how this can be violated is to suppose that there are many edit and batch jobs in the queue. It follows from the way jobs are scheduled that the density of edit jobs is higher toward the head of the queue and that the density of batch jobs is greater toward the queue's tail. Another way to view this situation is that, in the steady state, edit jobs are assumed to have a smaller wait for processing than do batch jobs; i.e., they are given smaller deadlines. Thus they tend, upon entry, to be placed in the queue ahead of many batch jobs. Now suppose that the expansion factor increases because of an increased system load. Then newly arriving edit jobs might, as outlined above, be placed behind batch jobs already in the queue. Since jobs already in the queue do not change their relative positions, edit jobs toward the head of the queue are the first to receive processing and leave the system. Newly arriving edit jobs, however, do not replace those that finish execution; thus, there is a period during which the density of edit jobs toward the head of the queue decreases. The CPU, during this period, processes an increased density of batch jobs; thus, the proportion of CPU time allocated to edit jobs decreases. The result is a substantial increase in delay for the newly arrived edit jobs.)

- *Derivation of the correspondence*

Section 3 discussed the correspondence between deadline scheduling and delay-cost minimization. We now make these arguments precise by deriving a corresponding DCR version of a DS. In the following subsection, we compare the performance of these two schedulers through simulation. As previously mentioned, the most notable result arising from this correspondence is a stable form of an analogous DS. Our procedure for deriving the correspondence is to obtain the delay-cost analogies of the quantities $\mathbb{E}$ and H_k given in Equation (8), and then to demonstrate how these values change under conditions of varying load. We first describe our notation and assumptions.

For a given class k, we assume that arrivals come from a Poisson point process with an average rate of λ_k. The service time for each class is assumed to be generally distributed with a mean of $\bar{x}_k$, and the processor utilization due to the kth class, denoted by ρ_k, equals $\lambda_k \bar{x}_k$. We define

$$\rho = \sum_{k=1}^{K} \rho_k$$

and

$$W_0 = \sum_{k=1}^{K} \lambda_k E[x_k^2]/2.$$

The delay-cost function for the kth class is given by $C_k(t) = s_k t$, where $s_1 \geq s_2 \geq \cdots \geq s_K$, and we denote the value of the offsets for the DS by u_k, where $u_1 \geq u_2 \geq \cdots \geq u_K = 0$. Finally, we let $\bar{T}_k$ be the mean response time for jobs of class k.

We now cite two results that are needed to create the correspondence. The first is called the conservation law [4, 18], which states that

$$\sum_{k=1}^{K} \rho_k(\bar{T}_k - \bar{x}_k) = \frac{\rho W_0}{1 - \rho} \tag{10}$$

and holds for all scheduling disciplines that have Poisson arrivals, are work-conserving, and service jobs nonpreemptively. The second result concerns the mean class-k response time for a DS under the above assumptions and with a heavy load. This heavy-traffic response time was derived in [14] and is discussed here only briefly. As mentioned above, scheduling decisions are made on the basis of priorities that increase with unit slope (the expansion factor is assumed to be a constant, since the arrival rates are fixed). It is further assumed that the load is sufficiently high that

$$W_k \geq u_1 \qquad k = 1, 2, \cdots, K, \tag{11}$$

where W_k is the random variable representing the waiting time of the kth class. Then,

$$\bar{T}_k \simeq \frac{W_0}{1 - \rho} + \bar{x}_k + \left(\sum_{j=1}^{K} \rho_j u_j \right) - \rho u_k \qquad k = 1, 2, \cdots, K. \tag{12}$$

To lend intuition to this equation, we note that $W_0/(1 - \rho) + \bar{x}_k$ is the mean response time that would be obtained for class-k jobs if FCFS scheduling were used [19]. The remaining two terms represent an adjustment of response time of the kth class due to priority scheduling. We write them as

$$\sum_{j=1}^{K} \rho_j (u_j - u_k).$$

To explain this sum, we tag an arrival at time t_0 of a class-2 job in a three-class system. Since by assumption $W_2 \geq u_1$, any class-1 jobs arriving in the interval $(t_0, t_0 + u_1 - u_2)$ receive service before the tagged job. Since each class-1 job requires an average of $\bar{x}_1$ seconds of work, and the jobs arrive at a rate of λ_1, the expected

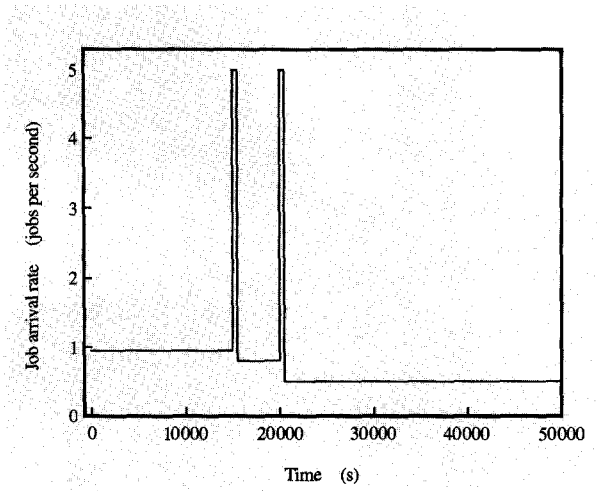

Figure 3

Arrival rate of jobs in the simulations, as a function of time.

amount of class-1 work that arrives during this interval is given by $\rho_1(u_1 - u_2) \geq 0$, which is the first term of the sum. The second term, $\rho_1(u_2 - u_2)$, is 0. The third term, $\rho_3(u_3 - u_2) \leq 0$, represents the amount of class-3 work already in the system at time t_0 that receives service after the tagged job. Since these jobs do not delay the tagged job, this amount of work contributes negatively to its response time in relation to FCFS scheduling.

We now proceed to establish the correspondence. We again use results obtained under conditions of heavy traffic and assume Poisson arrivals. Thus, the correspondence holds only approximately for Poisson arrivals and light to moderate load conditions, and becomes more precise as the load increases. Let $\bar{T}_k$ and $\bar{V}_k$ denote the expected class response times for the DCR scheduler and DS, respectively. The cost function $C(\cdot)$ for the DCR scheduler is linear, with slope s_k and zero offsets, and the deadline curves are linear with slope 1 and offsets u_k. In determining the deadline at a particular instant, one must specify values for $\mathbb{E}$, the expansion factor, and H_k. As shown in Section 3, for a given value of $\mathbb{E}$, the values of H_k determine values for the offsets u_k used in the DS. Our procedure consists of first establishing for a DCR scheduler a value that is analogous to the expansion factor $\mathbb{E}$. This value, as shown in Section 3, is given by β. This determines values for the mean response time, $\bar{T}_k$, for a DCR scheduler. These values can then be used in Equation (12) to determine values of u_k that would cause a DS to achieve $\bar{V}_k = \bar{T}_k$. The last step in establishing the correspondence consists of showing how the values of u_k change with changing load conditions and of specifying how to measure $\mathbb{E}$. We now present these derivations.

1. Derivation of β.
   Proposition 3 of the Appendix indicates that, under heavy load, there exists a β such that a DCR scheduler with delay-cost equations given by $C_k(t) = s_k t$ has mean response times that satisfy

$$\frac{s_k \bar{T}_k}{\bar{x}_k} = \beta. \tag{13}$$

   Use of this in the conservation law, Equation (10), yields

$$\beta = \frac{\dfrac{\rho W_0}{1-\rho} + \sum_{k=1}^{K} \rho_k \bar{x}_k}{\sum_{k=1}^{K} \dfrac{\rho_k \bar{x}_k}{s_k}}. \tag{14}$$

2. Calculation of u_k.
   Using β from Equation (13) allows us to write $\bar{T}_k = \beta \bar{x}_k / s_k$, $k = 1, 2, \cdots, K$. Given these mean response times, we can set $\bar{V}_k = \bar{T}_k$ by using the linear equations (12) to solve for the offsets u_k. For a given load, this yields a DS that has mean response times equal to what one would find in a DCR scheduler under heavy traffic.
3. Changing u_k and β.
   We now show how values of the offsets are modified to account for changing load conditions. First, note that substituting from Equation (13) into (9) yields

$$\mathbb{E} = \sum_{k=1}^{K} \frac{a_k \bar{T}_k}{\bar{x}_k} = \beta \sum_{k=1}^{K} \frac{a_k}{s_k}. \tag{15}$$

   Suppose now that the DCR scheduler load increases to β', where $\beta' = \gamma\beta$. It thus follows from Equation (13) that $\bar{T}'_k = \gamma \bar{T}_k$; i.e., the expected response times also increase by a factor of γ for a DCR scheduler.

   To show that the DCR scheduler and the DS have equivalent stationary response times, we first observe from Proposition 3 in the Appendix that

$$\bar{V}_k = U - u_k, \tag{16}$$

   where U is a class-independent constant. From the matching of expected class response times of the DCR scheduler and DSs in step 2 above, we have that $U = u_k + \beta \bar{x}_k / s_k$. We now ascertain for what values of u'_k we obtain $\bar{V}'_k = \bar{T}'_k$. To answer this, observe that if $\bar{V}'_k = \bar{T}'_k$, Equation (15) implies that the expansion factor changes by a factor of γ, i.e., that $\mathbb{E}' = \gamma\mathbb{E}$, which implies that the offsets of the DS also change by a factor of γ (i.e., that $u'_k = \mathbb{E}' H_k = \gamma u_k$). This, however, implies that $U' = u'_k + \beta' \bar{x}_k / s_k = \gamma U$. Thus, Equation (16) shows that the expected class response time for the DS is $\bar{V}'_k = U' - u'_k = \gamma \bar{V}_k$. This, as in the DCR scheduler, also increases by a factor of γ.

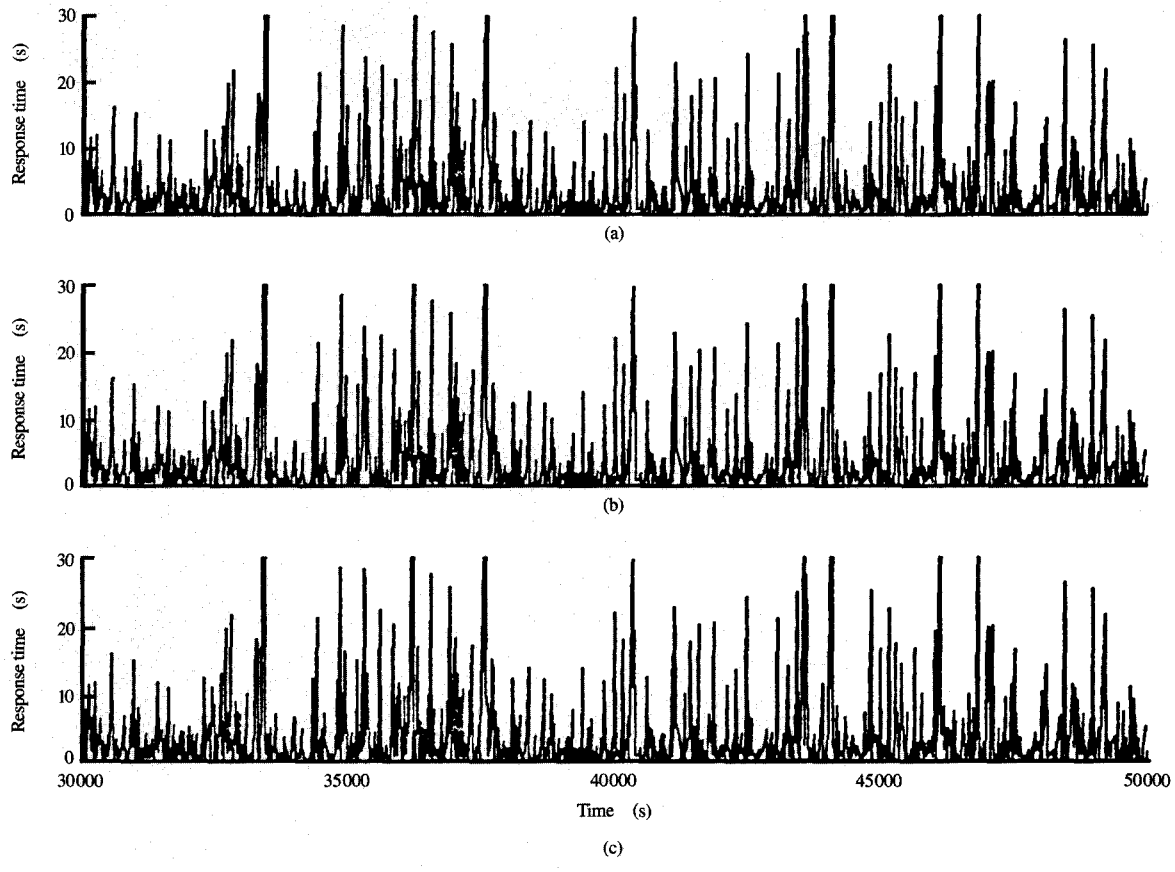

Figure 4

Class-1 response times during steady state, from simulation: (a) deadline scheduler with sampling period $\Delta t = 100$ s; (b) deadline scheduler with $\Delta t = 1000$ s; (c) delay-cost-ratio scheduler.

Since the values of u_k' needed to obtain $\bar{V}_k' = \bar{T}_k'$ are exactly those used by the DS, we conclude that the DS acts in a manner that matches the expected response of a DCR scheduler with linear slopes.

The correspondence is thus specified. We now compare the performance of these two schedulers via simulation.

- *Comparison by simulation*

This section presents simulation results that compare the performance of a DS and its corresponding DCR scheduler. We use the procedure presented above to establish the corresponding schedulers. Experiments are described that compare the performance obtained from DSs with their corresponding DCR schedulers. Each experiment consisted of processing the same arrival stream of jobs, starting from an initially empty system. Two different DSs were simulated. These differ only in the duration of the sampling period used in calculating the deadline, Δt, and correspond to an underdamped scheduler ($\Delta t = 100$ s) and a moderately damped scheduler ($\Delta t = 1000$ s). The third simulation is that of a DCR scheduler.

We first provide the input parameters used to drive the simulations. Three classes of jobs were assumed, with delay-cost curves given by $C_k(t) = s_k t$, where $s_1 = 100$, $s_2 = 10$, and $s_3 = 1$. The average service requirements were set to $\bar{x}_1 = 1$ s, $\bar{x}_2 = 7$ s, and $\bar{x}_3 = 8$ s. For any arrival rate of jobs, we set the fraction of arrivals of jobs in the three classes to be 60%, 17%, and 23%, respectively. This corresponds to a relative system utilization of approximately 1:2:3 for the three classes. The arrival rate during the simulations, shown in **Figure 3**, was nonstationary. The system utilization was 0.95 for $0 \le t \le 15000$, 0.80 for $15500 < t \le 20000$, and 0.50 for $20500 < t \le 50000$. The average utilization

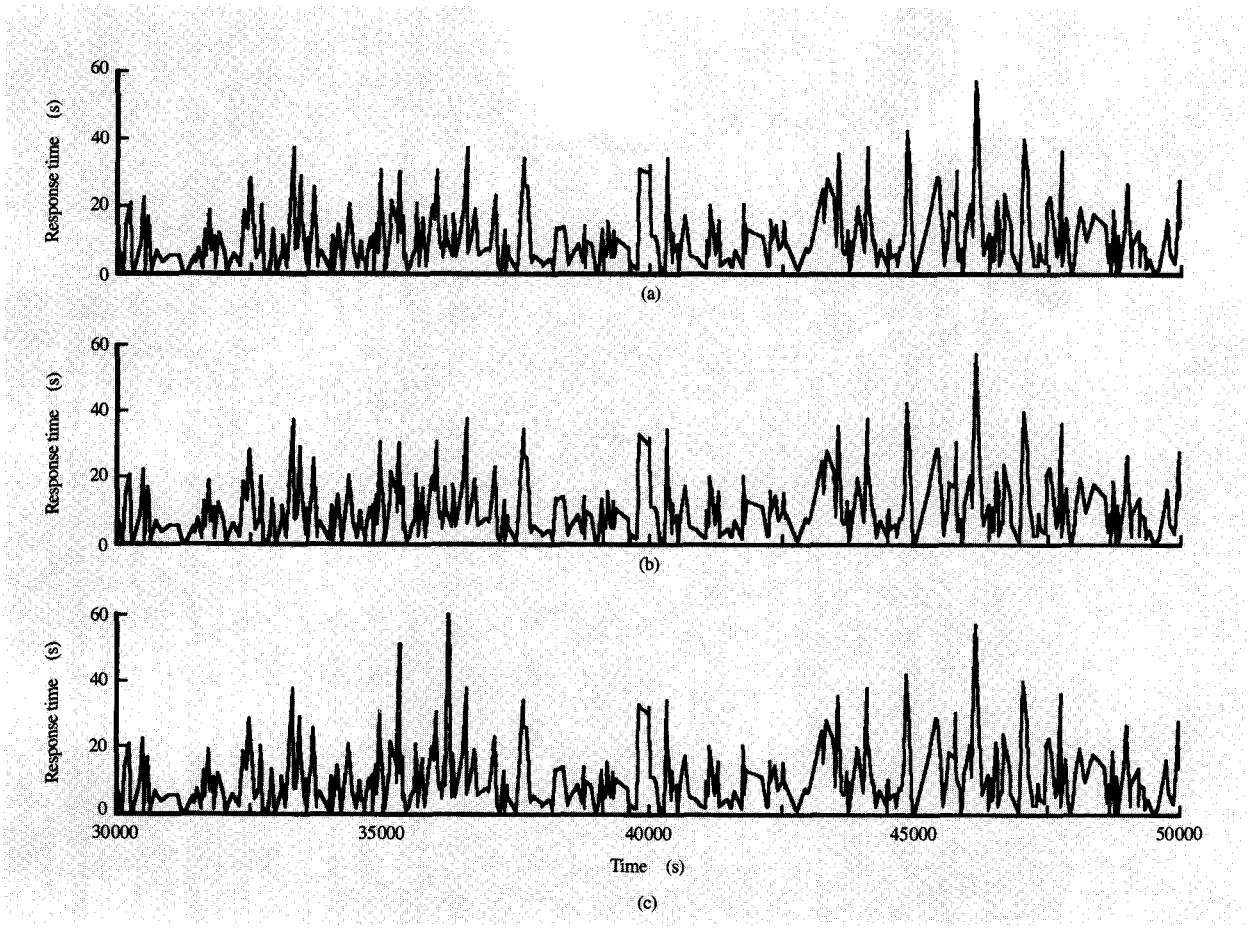

Figure 5

Class-2 response times, from simulation: (a)–(c) as in Figure 4.

over the entire interval $0 \leq t \leq 50000$ was 0.68. Note that there are high impulses of traffic over the intervals $15000 \leq t \leq 15500$ and $20000 \leq t < 20500$.

We now present the simulation results, which indicate that the DS and the corresponding DCR scheduler make similar scheduling decisions under conditions of steady state and constant load. This lends support to the equations used to derive the corresponding schedulers. We first note that the system was not in steady state over the interval $0 \leq t \leq 20500$. This follows from the fact that over this interval, the arrival rate varied with time and that the system started from an initially empty state. Empirically, we found that the system reached steady state at time 30000. We present results of the simulation over this interval to confirm that the deadline and DCR schedulers are analogous.

Figure 4 shows the response time as a function of the time of departure of each class-1 job over the interval $30000 \leq t \leq 50000$. The system is in steady state, and the load is constant. The deadline calculations for both DSs are thus similar and yield similar scheduling decisions. Figure 4(c) shows the response for class-1 jobs for the DCR scheduler. A comparison with Figures 4(a) and 4(b) demonstrates that, for class-1 jobs, the DCR scheduler makes scheduling decisions similar to those made by the DSs. The corresponding response times of class-2 and class-3 jobs are shown in **Figures 5** and **6**, respectively, and demonstrate that scheduling decisions for these job classes are similar for the different schedulers. (Note that Figures 4, 5, and 6 have different vertical scales.)

Next, in **Figure 7,** we plot the threshold values of class-1 jobs (the values of $\mathbb{E}H_1$) as a function of time, for each DS experiment. The underdamped deadline scheduler exhibits the oscillations that were anticipated above.

To see how the oscillations in the threshold calculation influence the response times of jobs in the system, we

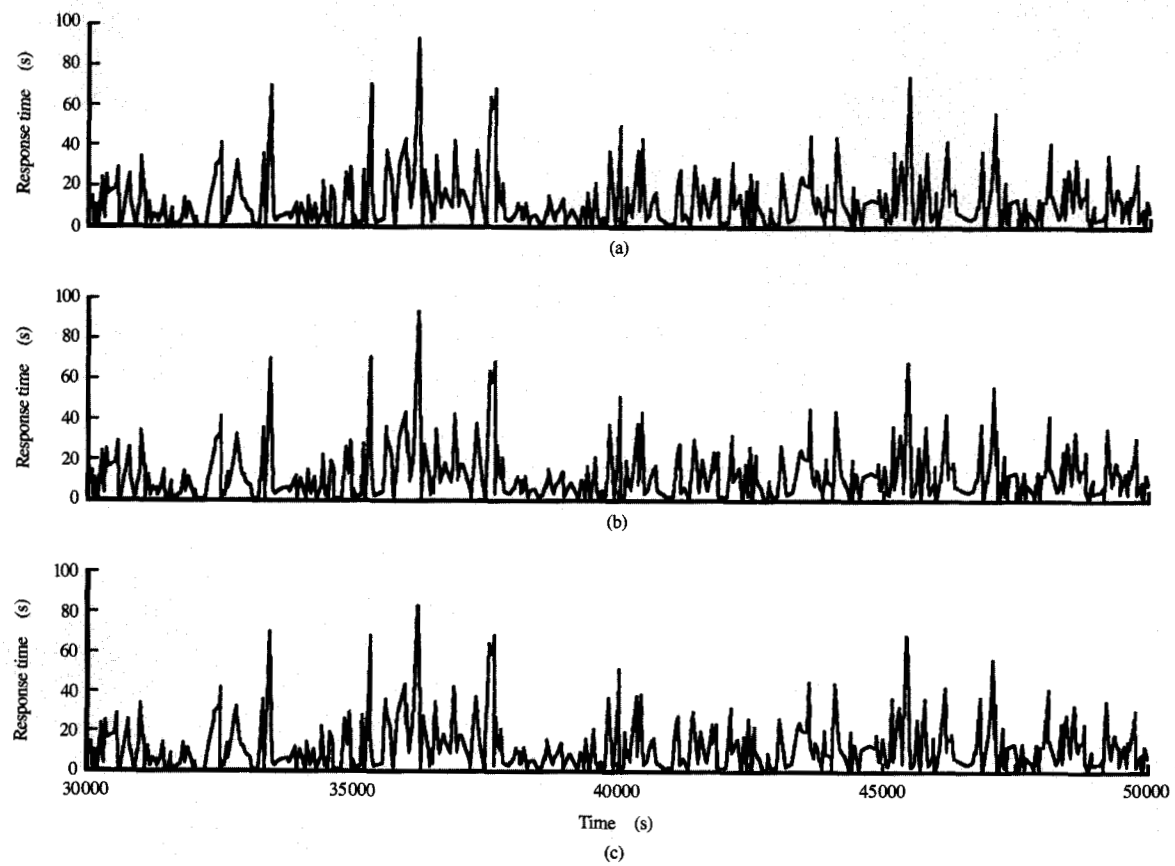

Figure 6

Class-3 response times, from simulation: (a)–(c) as in Figure 4.

plot, as a function of the time the job left the system, the response times for the different classes of jobs. **Figure 8** shows the response time for class-1 jobs for each of the schedulers. (Note that Figure 4 presents some of the same data, for the interval $30000 \leq t \leq 50000$, but with a different vertical scale.) Observe that in both deadline schedulers there are sudden increases in the response time near the points $t = 15000$ and 20000, because of the increased traffic. Intuitively, we realize that the impulses of traffic at these points drastically increase the thresholds and cause newly arriving class-1 jobs to be placed behind already existing class-2 and class-3 jobs in the queue. These class-1 jobs suffer long delays. The DCR scheduler adapts to the changing load without degrading the response times of class-1 jobs. The total delay cost for processing all of the jobs is given by 2.39×10^9, 3.12×10^9, and 1.01×10^9 for the underdamped DS, moderately damped DS, and DCR scheduler, respectively. Thus, the total delay cost is smallest for the DCR scheduler and largest for the moderately damped DS. Increasing the damping in the DS reduces undesirable oscillations but increases the total delay cost, since the scheduler cannot react as quickly to load variations.

We now concentrate on the interval (15000, 24000). **Figure 9** shows large oscillations in the response times of class-2 jobs over this interval for both DSs. In contrast, the DCR scheduler adapts to the changing load conditions by increasing the response times of class-2 jobs in a more gradual manner.

The oscillations in response time are seen most dramatically in **Figure 10**, which shows the statistics for class-3 jobs. Once again, in the DCR scheduler, class-3 job response times increase in a gradual manner. The interaction of the threshold and the response time of the jobs in the system can be seen by inspecting the response times of class-3 jobs over the interval

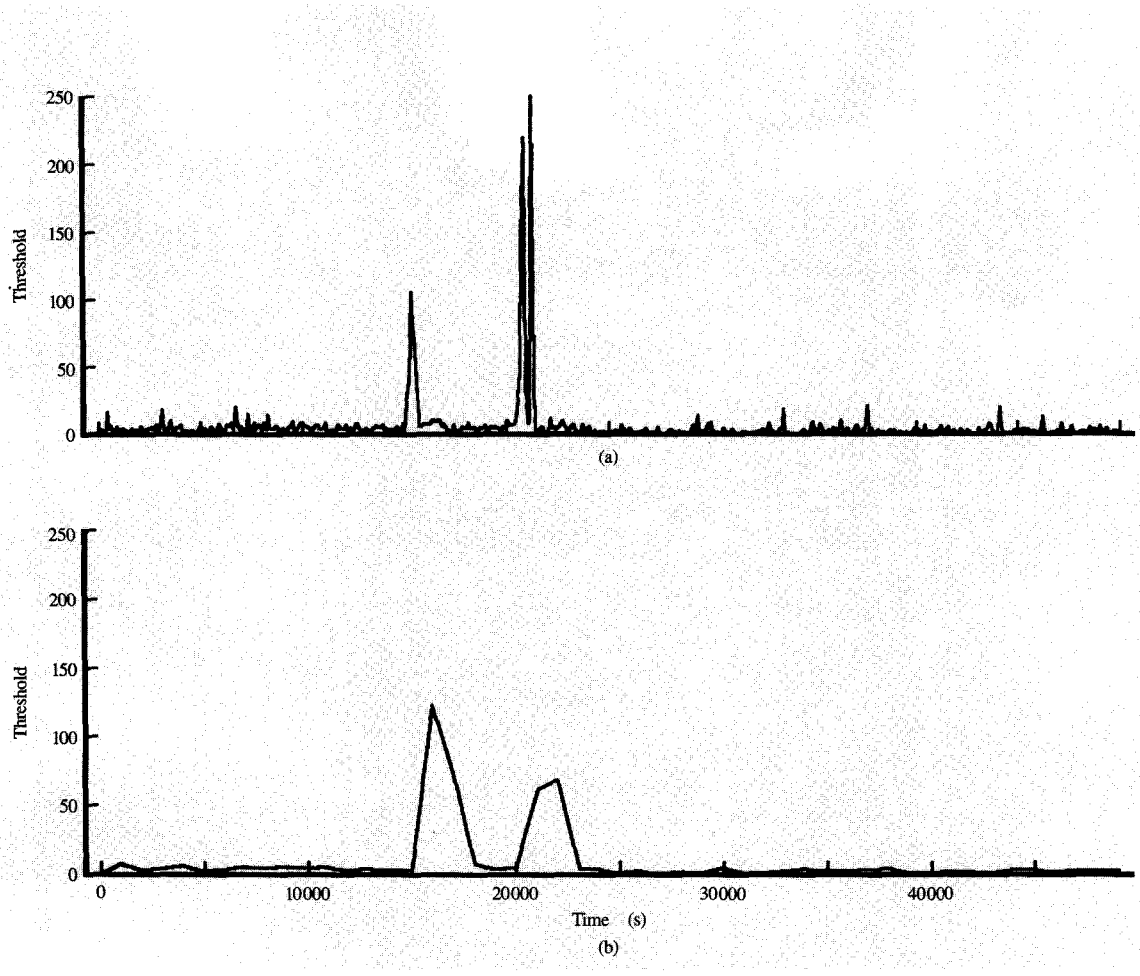

Figure 7

Threshold values for deadline schedulers, during entire simulation: (a) $\Delta t = 100$ s; (b) $\Delta t = 1000$ s.

$22000 \leq t \leq 23500$. The high peaks over the interval $23000 \leq t \leq 23500$ are from jobs that were placed far back in the queue because of the increase in traffic over the interval $15000 \leq t \leq 15500$. Peaks over the interval $22000 \leq t \leq 22500$ are from jobs that entered the system after the threshold increase due to increased traffic over $20000 \leq t \leq 20500$. Finally, the very low response times shown over the interval $22000 \leq t \leq 23500$ are from those class-3 jobs that entered the system during the low-threshold period after time 22000. As this example shows, different jobs from a single job class, present in the system at one time, can have very different views of the load, since their positions in the queue reflect measured response times at their times of entry. The measured response time may have little relation to the response time actually experienced by a job.

5. Conclusions and future work

We have considered issues associated with the use of objective functions in the design of time-sharing schedulers. Assumptions underlying the delay-cost form have been made explicit, and we have shown that algorithms of the form used in some versions of VM and MVS, for example, can be viewed as heuristics for minimizing some form of this measure. Properties of optimal schedules for the delay-cost objective have been derived. The case of deadline schedulers has been treated in detail, and it has been shown that such schedulers have a potential stability problem when subjected to rapidly varying loads. Here, the implied delay-cost goal can be used to obtain algorithms that behave similarly under steady-state conditions, but that avoid instability.

An experimental delay-cost version of a VM scheduler has been implemented and used on a 3090 multiprocessor

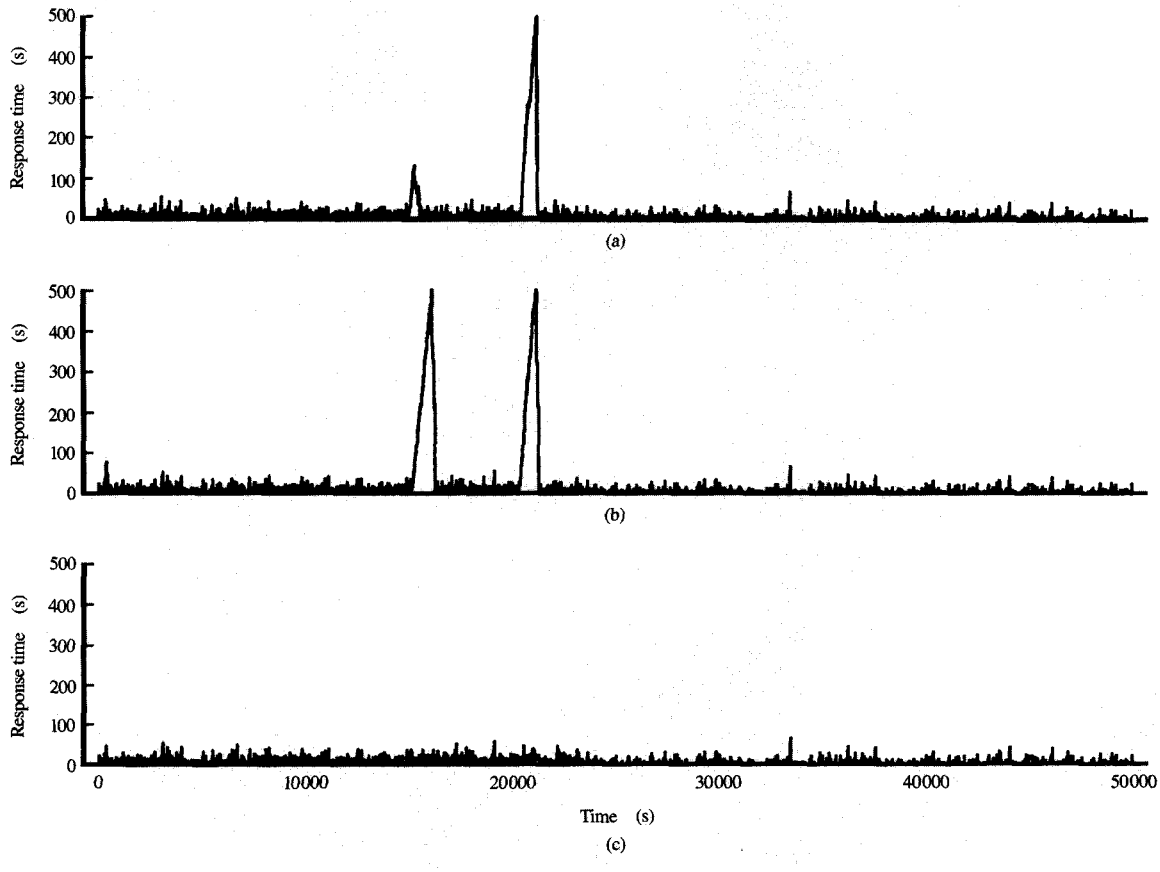

Figure 8

Class-1 response times, during entire simulation: (a) $\Delta t = 100$ s; (b) $\Delta t = 1000$ s; (c) DCR scheduler.

system at the IBM Thomas J. Watson Research Center.[2] Its main feature is the replacement of deadline-based algorithms with a delay-cost heuristic that uses a table-lookup computation based on the DCR algorithm. The remainder of the scheduler code, e.g., that used to estimate working sets, was not changed. Any piecewise-linear curve can be used as a delay-cost function: for example, a curve obtained from an interactive tool [20] that permits an installation manager to specify the number of job classes as well as requirements related to mean class service times. Experience gained with this system suggests that delay-cost-based scheduling is readily implementable, runs efficiently, and permits convenient specification of performance goals.

[2] The delay-cost version of the VM scheduler mentioned above was written by William Jerome and Gerald Spivak of the IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

The advantages of using a performance objective such as the delay cost include ease in specifying system goals and the possibility of considering a variety of scheduling algorithms in order, for example, to achieve stability under rapidly changing loads. Also of interest is the case of multiple processors in a system in which jobs have processor affinities. The latter problem is the subject of continuing work.

Appendix

This appendix provides justification for the properties of optimal delay-cost schedulers presented in Section 2. We restrict our attention to delay-cost functions that are increasing functions of time for all classes. This implies that it becomes incrementally more expensive for the system to keep a job the longer it stays in the system. We begin by presenting our definitions and assumptions.

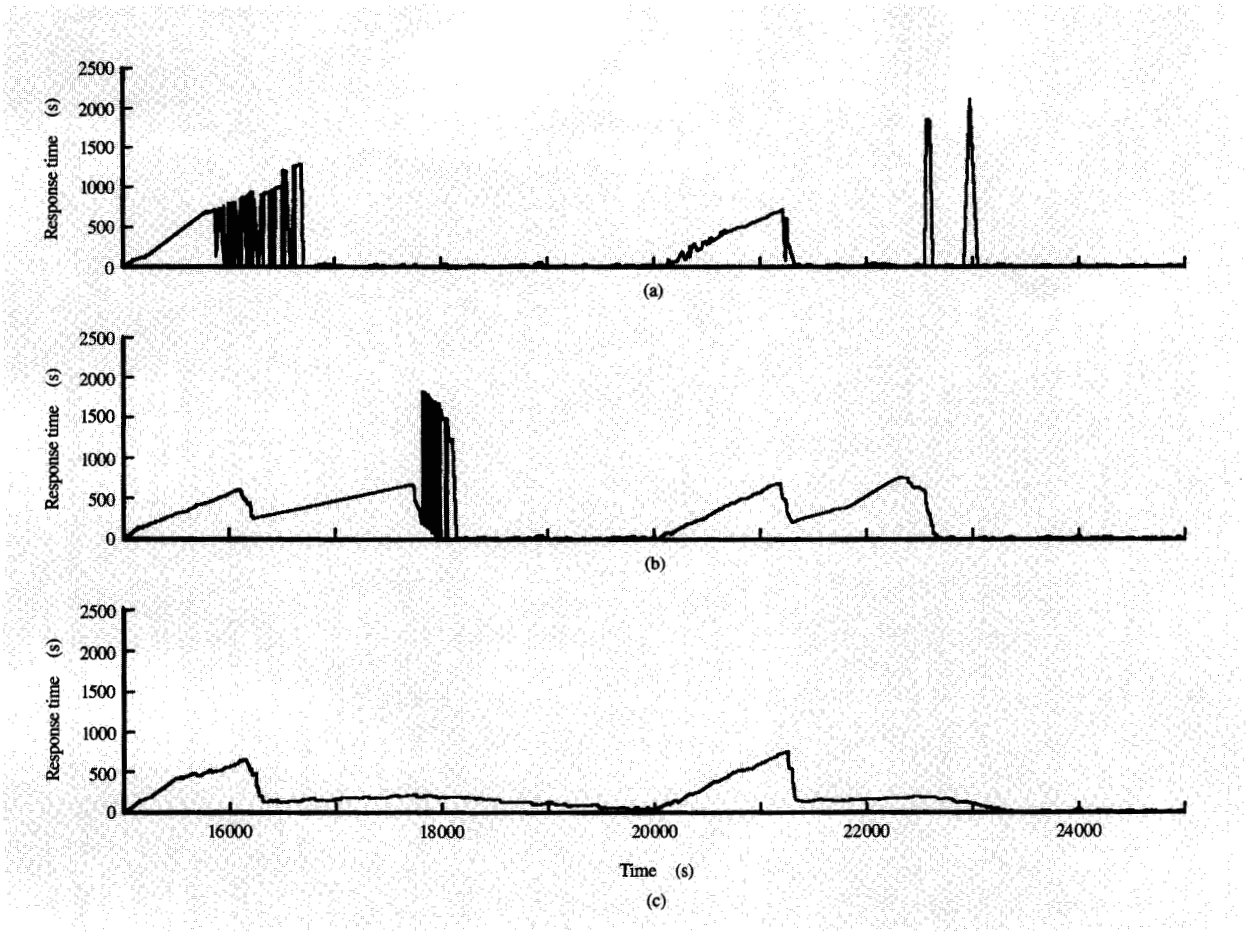

Figure 9

Class-2 response times, from simulation, for $15000 \leq t \leq 24000$: (a)–(c) as in Figure 8.

• *Assumptions and definitions*

A *busy period* begins when work arrives at an idle processor and ends when the processor again becomes idle. Since we consider only work-conserving policies, the busy periods for all policies are identical. We consider only ergodic systems; thus, busy periods are assumed to be finite with probability 1. If S^* is an optimal scheduling policy, then

$$F^{S^*}(J) = \min_S F^S(J), \tag{A1}$$

where $F^S(J)$ is given by Equation (5), and J is the sequence of jobs served during the busy period.

We use the following notation throughout this appendix: We assume that J is the set of jobs served in a given busy period and that job j, $j \in J$, has arrival and service times given by t_j and x_j, respectively. The service times for all classes are assumed to have finite moments of all orders. We let t be the current time and assume that the delay-cost functions are increasing and unbounded, and can be represented by finite-degree polynomials. For job i of class k, we let $D_i(t)$ be the delay-cost-ratio value (DCR) evaluated at time t, given by $D_i(t) = C_k(t - t_i)/\bar{x}_k$ if $t \geq t_i$ and equal to 0 otherwise. The algorithm that schedules the job having the maximal DCR value is called the DCR algorithm. We say that job i *dominates* job j from time t^* if $D_i(t) > D_j(t)$ for all $t \geq t^*$. Finally, we assume that scheduling policy S is an optimal policy.

• *Properties of optimal delay-cost schedulers*

Proposition 1

For each class, policy S^* executes the jobs in FCFS order.

Proof Let jobs i and i' be of class k and such that $t_i < t_{i'}$. Let $x_{j,k}$ be the service time for the jth scheduled class-k

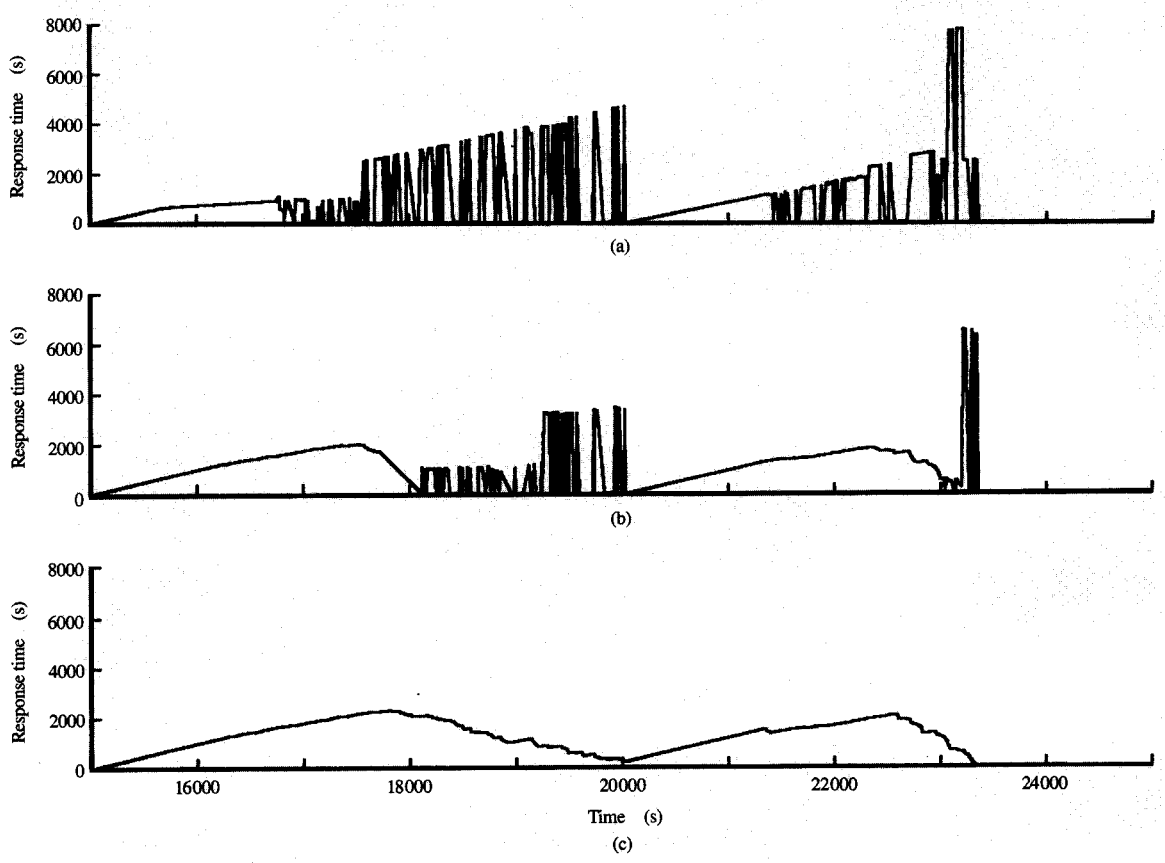

Figure 10

Class-3 response times, from simulation, for $15000 \leq t \leq 24000$: (a)–(c) as in Figure 8.

job. We assume that S^* schedules jobs within each class in an FCFS order, thus scheduling job i before i'. To show that this is optimal, define a policy S' that schedules jobs exactly as does S^*, except that it interchanges jobs i and i'. Suppose that job i is the jth class-k job scheduled by S^* at time t, and that job i' is the j'th, $j' > j$, class-k job, scheduled at time $t + x_{j,k} + z$, where $z \geq 0$. Policy S' thus schedules job i' at time t and job i at time $t + x_{j,k} + z$. The difference in the total costs for these two policies is given by

$$F^{S'}(J) - F^{S^*}(J) = E_{(x_{j,k}, x_{j',k})}\left[\int_{t-t_i+x_{j,k}}^{t-t_i+x_{j,k}+z+x_{j',k}} C_k(y)\,dy - \int_{t-t_{i'}+x_{j,k}}^{t-t_{i'}+x_{j,k}+z+x_{j',k}} C_k(y)\,dy\right], \quad \text{(A2)}$$

where the limits represent differences in residence times. The expectation of Equation (A2) is positive, since $C_k(y)$ is increasing, $x_{j,k}$ and $x_{j',k}$ have the same distribution, and $t_i < t_{i'}$. Thus, $F^{S'}(J) > F^{S^*}(J)$, so that FCFS is optimal. ■

We now establish a lemma that is used in the next proposition.

Lemma 1

Suppose that S' makes scheduling decisions identical to those of S' except as follows: S^* schedules job i of class k immediately before i' of class k', and S' schedules job i' immediately before i. We have

$$F^{S^*}(J) - F^{S'}(J) = E_{(t,x_j,j\geq 1)}\left\{\delta^{S^*}\left[\int_{t-t_{i'}+x_{i'}}^{t-t_{i'}+x_{i'}+x_i} C_{k'}(y)\,dy - \int_{t-t_i+x_i}^{t-t_i+x_i+x_{i'}} C_k(y)\,dy\right]\right\}, \quad \text{(A3)}$$

where δ^{S^*} is an indicator random variable that is equal to 1 if S^* schedules job i immediately before i' at a time t, for $t > \{t_i, t_{i'}\}$, and

$$t = \sum_{j \in J_i^{S^*}} x_j,$$

where $J_i^{S^*}$ is the set of jobs scheduled by S^* before job i.

Proof We can write $F^{S^*}(J) - F^{S'}(J) = F^{S^*}(J - \{i, i'\}) - F^{S'}(J - \{i, i'\}) + F^{S^*}(\{i, i'\}) - F^{S'}(\{i, i'\})$. By definition, S^* and S' are such that the execution times of all jobs in $J - \{i, i'\}$ are identical. Consequently, $F^{S^*}(J - \{i, i'\}) = F^{S'}(J - \{i, i'\})$. The proof then follows, since Equation (A3) is the value of $F^{S^*}(\{i, i'\}) - F^{S'}(\{i, i'\})$. ■

Proposition 2 (Pairwise optimality)
Suppose that jobs i and i' are the oldest jobs of their respective classes, k and k'. Furthermore, suppose that these are the next two jobs that are scheduled by S^*. Then, for any $\varepsilon > 0$, there exists a t^* such that if $t > t^* > \max\{t_i, t_{i'}\}$ and $D_i(t) \geq (1 + \varepsilon)D_{i'}(t)$, then S^* schedules job i first.

Proof Consider a policy S' that makes decisions identical to S except that, under the conditions of the proposition, it reverses the order of jobs i and i'. If S schedules i first, it follows from the lemma that Proposition 2 can be restated as

$$F^{S^*}(J) - F^{S'}(J) = E_{(x_i, x_{i'})}\left\{\delta^{S^*}\left[\int_{t-t_{i'}+x_{i'}}^{t-t_{i'}+x_{i'}+x_i} C_{k'}(y)\,dy - \int_{t-t_i+x_i}^{t-t_i+x_i+x_{i'}} C_k(y)\,dy\right]\right\} \leq 0. \quad \text{(A4)}$$

In (A4) we have explicitly written the expectation in terms of the random variables x_i and $x_{i'}$. For Equation (A4) to be satisfied, it is sufficient, for any fixed values of t, t_i, and $t_{i'}$ satisfying the conditions of the proposition, that the following expression is satisfied:

$$\frac{E_{(x_{i'}, x_i)}\left[\int_{t-t_{i'}+x_{i'}}^{t-t_{i'}+x_{i'}+x_i} C_{k'}(y)\,dy\right]}{E_{(x_{i'}, x_i)}\left[\int_{t-t_i+x_i}^{t-t_i+x_i+x_{i'}} C_k(y)\,dy\right]} \leq 1. \quad \text{(A5)}$$

We rewrite (A5) using the identities

$$\int_{\alpha+\beta}^{\alpha+\gamma} C_{k'}(y)\,dy = \sum_{j=1}^{\infty} \frac{(\gamma^j - \beta^j)}{j!} C_{k'}^{(j-1)}(\alpha)$$

and

$$(x_i + x_{i'})^j - x_{i'}^j = \sum_{n=0}^{j-1} \binom{j}{n} x_{i'}^n x_i^{j-n}.$$

This implies that

$$\frac{\sum_{j=1}^{\infty}\left[\frac{C_{k'}^{(j-1)}(t - t_{i'})}{j!} \sum_{n=0}^{j-1} \binom{j}{n} \overline{x_{i'}^n x_i^{j-n}}\right]}{\sum_{j=1}^{\infty}\left[\frac{C_k^{(j-1)}(t - t_i)}{j!} \sum_{n=0}^{j-1} \binom{j}{n} \overline{x_i^n x_{i'}^{j-n}}\right]} \leq 1. \quad \text{(A6)}$$

The interchange of summation and expectation in (A6) follows from the fact that the polynomials C_k and $C_{k'}$ are of finite order; thus, each summation has a finite number of nonzero terms. We can write (A6) in the form

$$\frac{D_{i'}(t)[1 + G_{i'}(t)]}{D_i(t)[1 + G_i(t)]} \leq 1, \quad \text{(A7)}$$

where

$$G_{i'}(t) = \frac{1}{D_{i'}(t)} \sum_{j=2}^{\infty}\left[\frac{C_{k'}^{(j-1)}(t - t_{i'})}{j!} \sum_{n=0}^{j-1} \binom{j}{n} \overline{x_{i'}^n x_i^{j-n}}\right]$$

and $G_i(t)$ is defined analogously. Since we assume all moments of the service time to be finite and the delay-cost functions to be polynomials of finite degree, both $G_i(t)$ and $G_{i'}(t)$ approach 0 as $t \to \infty$. Thus, for any $\varepsilon > 0$, there exists a t^* such that if $t > t^*$, then $[1 + G_{i'}(t)]/[1 + G_i(t)] \leq 1 + \varepsilon$. Using this in Equation (A7), with the assumption that $D_i(t) \geq (1 + \varepsilon)D_{i'}(t)$, implies that (A7), and thus (A5), is satisfied. This implies, however, that (A4) is satisfied, thus that S schedules i first. ■

Proposition 2 shows that for sufficiently high load, S will never schedule a job with a lower DCR value before a job with a higher DCR value. It does not imply, however, that the DCR algorithm is necessarily optimal. As a counter-example, consider the case in which job i of class k and job i' of class k' are the oldest jobs of their respective classes and that $D_i(t) < D_{i'}(t)$. Furthermore, assume that job i' does not dominate job i from time t, and assume that $D_i(t + \varepsilon) \gg D_{i'}(t + \varepsilon)$, where ε is a small number greater than 0. If these two jobs were to be executed sequentially, Proposition 2 implies that i' would be scheduled next. Suppose, however, that a batch of class-k jobs arrives within ε seconds of job i. These jobs have DCR values close to $D_i(t)$, and after ε seconds, the DCR values for these jobs are all much greater than the DCR value of job i'. It is easy to check that, for this case, it

is optimal to schedule several class-k jobs before finally scheduling i'. In such a schedule, at no time will like jobs be scheduled out of DCR order, but it is not necessarily the case that the job with the highest DCR value is the next to be scheduled.

The next proposition establishes a relationship between the mean class waiting times when jobs are scheduled according to a time-dependent priority scheduler for which the priorities are given by linear functions of the time in the system. Specifically, a job of class k that has been in the system t seconds has a current priority value given by a linear function $p_k(t)$, for $t \geq 0$. We assume that $p_1(t) \geq p_2(t) \geq \cdots \geq p_K(t)$ for $t \geq 0$, that arrival rates are Poisson and class-dependent, and that service times are class-dependent, generally-distributed random variables with finite mean. The total utilization of the system is denoted by ρ.

Proposition 3
In the limit of heavy traffic, as $\rho \to 1$, the following equation is satisfied by all job classes:

$$p_k(\bar{T}_k) = \beta(\rho), \tag{A8}$$

where $\beta(\rho)$ is an increasing function of ρ. Proposition 3 is proved in [14]. Note that for linear delay-cost curves with zero offsets, i.e., $C_k(t) = s_k t$, the proposition shows that the DCR algorithm satisfies $C_k(\bar{T}_k)/\bar{x}_k = \beta$ in heavy load.

Acknowledgment

The authors thank Stephen Lavenberg and Joseph Hellerstein for their insightful criticisms of this work and for their careful reading of the manuscript.

References

1. M. Greenberger, "The Priority Problem and Computer Time Sharing," *Manage. Sci.* **12,** No. 11, 888–906 (July 1966).
2. *IBM Virtual Machine/System Product High Performance Option*, Program Number 5664-173, 5664-167, 1983.
3. E. Gelenbe and I. Mitrani, *Analysis and Synthesis of Computer Systems*, Academic Press, Inc., New York, 1980.
4. E. G. Coffman and I. Mitrani, "A Characterization of Waiting Time Performance Realizable by Single-Server Queues," *Oper. Res.* **28,** No. 3, 810–821 (1980).
5. I. Mitrani and J. H. Hine, "Complete Parameterized Families of Job Scheduling Strategies," *Acta Inform.* **8,** 61–73 (1977).
6. R. W. Wolff, "Time Sharing with Priorities," *SIAM J. Appl. Math.* **19,** 566–574 (1970).
7. L. Kleinrock, *Queueing Systems Volume 2: Computer Applications*, John Wiley, Inc., New York, 1975.
8. G. P. Klimov, "Time Sharing Service Systems. I," *Theor. Prob. & Appl.* **XIX,** No. 3, 532–551 (1974).
9. G. P. Klimov, "Time Sharing Service Systems. II," *Theor. Prob. & Appl.* **XXIII,** No. 2, 314–321 (1978).
10. Z. Rosberg, "Process Scheduling in a Computer System," *IEEE Trans. Computers* **C-34,** No. 7, 633–644 (1985).
11. L. Kleinrock, "A Delay Dependent Queue Discipline," *Nav. Res. Log. Quart.* **9,** 31–36 (1962).
12. L. Kleinrock and R. P. Finkelstein, "Time Dependent Priority Queues," *Oper. Res.* **15,** 104–116 (1967).
13. A. Netterman and I. Adiri, "A Dynamic Priority Queue with General Concave Priority Functions," *Oper. Res.* **27,** 1088–1100 (1979).
14. R. Nelson, "Heavy Traffic Response Times for a Priority Queue with Linear Priorities," *Oper. Res.* **38,** 560–563 (1990).
15. U. Bagchi and R. Sullivan, "Dynamic, Non-Preemptive Priority Queues with General, Linearly Increasing Priority Function," *Oper. Res.* **33,** 1278–1298 (1985).
16. P. Franaszek and R. Nelson, "Bounds for Scheduling Timeshared Parallel Processors," *Research Report RC-16314*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, September 1990.
17. J. E. Gibson, *Nonlinear Automatic Control*, McGraw-Hill Book Co., Inc., New York, 1963.
18. L. Kleinrock, "A Conservation Law for a Wide Class of Queueing Disciplines," *Nav. Res. Log. Quart.* **12,** 181–192 (1965).
19. L. Kleinrock, *Queueing Systems Volume 1: Theory*, John Wiley, Inc., New York, 1975.
20. P. Franaszek and R. Nelson, "A Tool for Creating Delay Cost Curves for a DCR Scheduler," *Research Report RC-14184*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, November 1988.

Received February 15, 1993; accepted for publication April 22, 1994

Peter A. Franaszek *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (PAF at YKTVMV, paf@watson.ibm.com).*

Dr. Franaszek is manager of Systems Theory and Analysis in the Computer Sciences Department at the Thomas J. Watson Research Center. He received an Sc.B. degree from Brown University in 1962, and M.A. and Ph.D. degrees from Princeton University in 1964 and 1965, respectively. Dr. Franaszek's interests include analytical and design issues in computer system organization, algorithms, and communication networks and coding. He has received IBM Outstanding Innovation Awards for his work in the areas of algorithms, interconnection networks, concurrency control theory, and constrained coding. Dr. Franaszek was also the recipient of two IBM Corporate Awards for his work in the latter area. In 1991, he was elected to the IBM Academy of Technology. He was named the recipient of the 1989 Emanuel R. Piori Award of the Institute of Electrical and Electronics Engineers for his contribution to the theory and practice of digital recording codes. During the academic year 1973–1974 he was on sabbatical leave from the Thomas J. Watson Research Center to Stanford University as a Consulting Associate Professor of Computer Science and Electrical Engineering. Prior to joining IBM in 1968, he was a member of the technical staff at Bell Telephone Laboratories. Dr. Franaszek is a member of Tau Beta Pi and Sigma Xi, and a Fellow of the IEEE.

Randolph D. Nelson *OTA Limited Partnership, One Manhattanville Road, Purchase, New York 10577 (rnelson@oc.com).* Dr. Nelson received his Ph.D. from UCLA in 1982 and joined the IBM Thomas J. Watson Research Center in Yorktown Heights in the Systems Analysis Department in the same year. His initial research interests at the lab included applied probability and computer-performance modeling. From 1988 to 1993, he managed the performance modeling methodology group, an analytic research group engaged in basic research into new theoretical results and applications of probability, stochastic processes, and simulation. During 1993 he spent a sabbatical at the IBM Retirement Fund and used analytical techniques to create option-trading strategies. After leaving IBM in 1994, he joined OTA Limited Partnership, a hedge fund, as Vice President of Investment Research. His current work is concerned with mathematical techniques as they apply to financial trading strategies.