On-Line Scheduling Policies for a Class of IRIS (*Increasing Reward with Increasing Service*) Real-Time Tasks

Jayanta K. Dey, James Kurose, *Member, IEEE*, and Don Towsley, *Fellow, IEEE*

Abstract—We consider a real-time task model where a task receives a "reward" that depends on the amount of service received prior to its deadline. The reward of the task is assumed to be an increasing function of the amount of service that it receives, i.e., the task has the property that it receives *increasing reward with increasing service (IRIS)*. We focus on the problem of on-line scheduling of a random arrival sequence of IRIS tasks on a single processor with the goal of maximizing the average reward accrued per task and per unit time. We describe and evaluate several policies for this system through simulation and through a comparison with an unachievable upper bound. We observe that the best performance is exhibited by a two-level policy where the top-level algorithm is responsible for allocating the amount of service to tasks and the bottom-level algorithm, using the earliest deadline first (EDF) rule, is responsible for determining the order in which tasks are executed. Furthermore, the performance of this policy approaches the theoretical upper bound in many cases. We also show that the average number of preemptions of a task under this two-level policy is very small.

Index Terms—Real-time systems, on-line scheduling, deadline based scheduling, priority scheduling, reward functions for tasks, maximizing reward rates.

1 Introduction

In a classical hard real-time system, each task must complete execution before its specified deadline. Failure of the system to complete a task by its deadline leads to a timing-fault and the results of the computation (if any) are discarded [1]. Implicit in such systems is the assumption that a task's computation is useless unless the entire computation is completed within the specified deadline.

Recently, a new class of real-time applications has emerged in which the *value* of a task's computation increases as a function of the amount of execution time it has been able to accrue before its deadline expires. We refer such tasks as IRIS (Increasing Reward with Increasing Service) tasks, since the longer a task is able to execute (before its deadline), the higher is the quality of its computation. Tasks exhibiting IRIS behavior have been referred to in the literature as "anytime algorithms" [2], [3], [4], "real-time approximate processing algorithms" [5], or "segmented algorithms" [6]. Examples include tasks that receive, enhance or transmit audio, video or images compressed using MPEG-2 or other scalable techniques [7], [8], [9], [10], tasks for tracking and control, e.g., autonomous vehicle navigation planning [11], tasks for heuristic search [12], [13], tasks for database query processing [14], [15], medical decision making [16], and tasks computing traditional iterative-refinement numerical algorithms. Methods of construction of "anytime algorithms" as well as detailed examples of "anytime algorithms" are included in [17].

Within the framework of the IRIS model, we study the problem of scheduling IRIS tasks in a single processor environment, where IRIS tasks arrive into the system at arbitrary instances of time and provide the scheduler with their deadlines and their reward functions. The main contributions of this paper are:

- The design and evaluation of three *on-line* scheduling policies for the case that the task reward functions (i.e., functions which specify the value of a computation as a function of its accrued execution time) are nondecreasing and concave. The first two policies consist of a top-level algorithm that executes at task arrival times to determine the amount of service to be allocated to each task and a bottom-level algorithm that determines the order in which tasks execute. Both policies use the same top-level algorithm and differ only in the rule used to schedule tasks at the bottom level. The third policy is a (single level) greedy scheduling policy.

  We note that the top-level algorithm alluded to above is of independent interest as it solves the static problem of *maximizing* the reward rate for a set of tasks that arrive at the same time.
- The computation of a theoretical upper bound on the performance of *any* IRIS scheduling policy.

• *The authors are with the Department of Computer Science, University of Massachusetts, Amherst, MA 01003.*
E-mail: {dey, kurose, towsley}@cs.umass.edu.

Manuscript received June 28, 1994; revised Sept. 15, 1995.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C96111.

- The demonstration that the two-level policy that uses the earliest deadline first (EDF) rule for scheduling tasks into service achieves performance close to the upper bound and better than that of either of the other two policies. It is also shown that the average number of preemptions of a task under this two-level policy is small.

In order to place these results in their appropriate context within the very large body of past scheduling research, we first distinguish between job shop scheduling [18], [19] and on-line scheduling of jobs. In the former case, a fixed set of jobs must be scheduled. Typically, no new jobs arrive to the system, and each job has an execution time—the amount of time needed by a job until it is completed by the system. In the latter case, new jobs may continue to arrive (and hence must be scheduled) while the system is operating. Again, each job has an associated execution time. As noted earlier, real-time scheduling is further characterized by the fact that jobs have deadlines by which time their execution must be completed. Overviews of scheduling work that include discussion of real-time systems can be found in [1], [20], [21], [22].

Our present work falls broadly in the second category but differs from traditional on-line scheduling in that IRIS tasks do not have specific execution times. Instead, as noted above, a task simply receives whatever service time it can accrue before its deadline expires. The reward function of an IRIS task (informally, the "value" of the task as a function of its accrued execution time) is typically a nondecreasing concave function of its execution time., reflecting the decreasing marginal reward a task sees as it receives more service [2], [4], [16].

Several papers have reported studies on real-time task models similar to the IRIS model. The *imprecise computation* model has generated a large body of work [23], [24], [25]. In this model, tasks have a *mandatory* phase with a certain known processing time[1] which has to be completed before the task is of any "value," and an *optional* phase with a known processing time which can be left unfinished. Each task incurs an error equal to the amount of the optional phase that is left unfinished. Thus after the mandatory phase of the task is completed, the quality of each task is assumed to be a linear function of the amount of processing time that it receives. Furthermore, all tasks are characterized by identical "error rates." We show in Section 6 how our scheduling mechanism can easily be adapted to handle tasks with explicitly defined mandatory phases. As noted before, the IRIS model does not require explicit knowledge of processing times of tasks, whereas in the imprecise computation model [23], [24], [25], tasks have known processing times. Note that the metric in IRIS of maximizing reward rate is analogous to the performance metric of minimizing error in the imprecise computation model.

For time-dependent path planning of robots [3], [4], Boddy and Dean consider a static system where a fixed set of tasks have to be scheduled. The tasks exhibit IRIS behavior. In their model, the processing times of tasks are multiples of a fixed time slice, i.e., it is a discrete-time version of the IRIS task model. They design an off-line scheduling policy to schedule tasks in this static system. The complexity of their scheduling algorithm is inversely proportional to the size of the time slice, and hence its running time increases with the fineness of the time slice, whereas the complexity of the IRIS scheduler depends only upon the number of tasks.

In a different model of imprecise real-time systems, tasks do not have deadlines, but each task provides several versions, a *primary version,* which produces a precise result but requires the maximum execution time of all the versions, and several *alternate versions,* which provide poorer quality results but execute for shorter times. The scheduling policy switches from primary versions of tasks to alternate versions when the total number of tasks in the system exceeds a threshold. Kim and Towsley [26] study this model in the context of real-time message transmissions, while Chong and Zhao [27] and Zhao, Vrbsky, and Liu [28] analyze two-version scheduling disciplines for this model in uniprocessor and multiprocessor systems respectively. The optimal scheduling policy for such a uniprocessor system was shown to be of a threshold type in [29].

Finally, we note that the framework for scheduling and planning of manufacturing systems which has been proposed by Gershwin [30] in which decisions and events in a production systems are grouped into various levels of hierarchy depending on their characteristic time-scales. Our two-level policies fit into Gershwin's scheduling framework. However they do not correspond to any of the algorithms that he describes.

The remainder of this paper is organized as follows. Section 2 describes the IRIS uniprocessor task system along with some attendant notation. Section 3 describes the proposed scheduling policies in detail. Section 4 develops an analytical model of IRIS tasks executing on a single processor and presents the upper bound on any scheduling policy. Section 5 presents the numerical results. Section 6 discusses extensions to the model to handle tasks with mandatory phases and Section 7 concludes the paper with a discussion of possible future research.

2 System Model

In this section, we describe the IRIS task model and the system on which they execute.

We consider a single processor system servicing a stream of tasks that arrive to the system at arbitrary times $a_1 \leq a_2 \leq \ldots$. Consider the ith task, $i = 1, 2, \ldots$. Associated with it is a laxity $t_i > 0$ and a nondecreasing concave reward function $f_i : \mathbb{R}_+ \rightarrow \mathbb{R}$. This task accrues a reward of $f_i(x)$ upon receiving x units of service and is required to leave after its laxity expires, at time $a_i + t_i$, referred to as its deadline.

The tasks present in the system are served according to some scheduling policy π. The only assumption made regarding π is that it is *nonanticipative,* i.e., it does not have knowledge of future arrival times or of the deadlines associated with future tasks. Otherwise, it is permitted to take any action including idling the processor or preempting tasks that are in service.

1. Here *known processing time* of a task implies that when the task arrives to the system, its processing time is explicitly known.

Under π, task i receives A_i^π units of service and accrues a reward of $f_i(A_i^\pi)$. The performance metric of interest to us is the long term reward rate given by

$$\gamma^\pi = \liminf_{t\to\infty} \frac{\sum_{i=1}^{N_t} f_i(A_i^\pi)}{t},$$

where N_t is the number of arrivals by time $t > 0$.

3 Scheduling Policies

The problem of determining a good scheduling policy for an online IRIS task system is difficult. Even in the case when tasks do not arrive dynamically and there is a finite number of tasks in the system at time $t = 0$ with arbitrary deadlines it is easy to show that the problem of determining an optimal policy for maximizing the total reward is, in general, NP-hard. In this section, we propose heuristic policies which provided demonstrably good performance. First, we propose a class of two-level scheduling policies. Each policy consists of a top-level algorithm which is executed each time that a task arrives and is used to determine the amount of service to allocate to tasks, and a lower-level algorithm that is concerned with the actual selection of tasks to execute. Policies within this class use the same high-level algorithm but differ in the particular rule for selecting tasks for service. We also consider a single level greedy policy, which is derived based on intuitions gained during the design of the top-level algorithm used by the two-level policies.

3.1 Two-Level Scheduling Approach

The top-level scheduling algorithm is invoked at every task arrival. It allocates service time to tasks with the objective of solving the following *static nonlinear optimization* problem: Given a set of tasks resident in the system with known deadlines and reward functions, determine the amount of service to allocate to each of them so as to maximize the total accrued reward. In a static scenario, where tasks do not arrive to the system dynamically, maximizing total accrued reward is equivalent to maximizing the reward rate. In this implementation, the algorithm that solves this problem executes under the assumption that there are no future arrivals. This static problem and several of its properties are described in Section 3.1.1 and its solution algorithms are presented in Section 3.1.2.

The lower-level algorithm actually schedules the tasks into service and attempts to provide them with the allocations determined by the top-level scheduler. However, the amount of execution time that each of the tasks actually receives depends upon the time of the next task arrival. Note also that the target service allocation may change at each arrival. We have experimented with the earliest-deadline-first algorithm and the first-come-first-serve algorithm as the lower-level policy. We discuss these in Section 3.1.3.

3.1.1 Formulation of the Problem with a Static Set of Tasks

In this subsection, we develop a static version of the problem where all tasks arrive at the same time. While such a simultaneous-arrival assumption is in itself restrictive, it can be used as the top-level computation of the two-level scheduling approach described above.

Consider a finite set of M tasks with identical arrival times $a_1 = a_2 = \ldots = a_M = a$ and deadlines $\tau'_1 \le \ldots \le \tau'_M$. For notational convenience, we define $\tau'_0 \equiv a$. The interval $(a, \tau'_M]$ is divided into at most M intervals $(a, \tau'_1], (\tau'_1, \tau'_2], \ldots, (\tau'_{M-1}, \tau'_M]$. As shown in Fig. 1, *interval* j refers to the interval $(\tau'_{j-1}, \tau'_j]$. When the ith task departs, if it has received x units of service, it receives a *reward* of $f_i(x)$, with its derivative defined to be $g_i(x) \equiv df_i(x)/dx$. The inverse function of $g_i(x)$ is $g_i^{-1}(x)$, $i = 1, \ldots, M$. The reward functions are assumed to be continuous and g^{-1} is assumed to be a function in the rest of this paper.

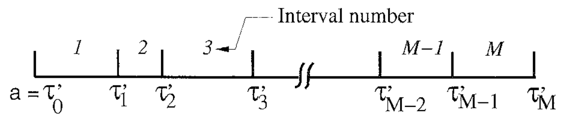


Fig. 1. A time interval showing task deadlines and interval numbers.

A *schedule* determines the amount of service to be given to each task during each interval. A *schedulable interval* of a task is an interval in which the task can be scheduled, i.e., it is an interval between the task's arrival time and its deadline. Thus the schedulable intervals of task i are intervals 1 through i. The set of *schedulable tasks* in an interval j is the set of all tasks $\{j, j+1, \ldots, M\}$ which can be scheduled in interval j.

Let $x_{i,j}$ denote the service time allocated to task i during interval j, $j = 1, \ldots, i$, $i = 1, \ldots, M$. Observe that $x_{i,j} = 0$ whenever $j > i$. The total amount of service allotted to task i is defined by

$$x_i = \sum_{j=1}^{i} x_{i,j}.$$

Given a set of tasks, we are interested in finding an assignment of service times to tasks which maximizes the sum of the rewards of all the tasks while satisfying the above constraints. Thus our problem, $\boldsymbol{P}$, is:

$$\text{maximize } \sum_{i=1}^{M} f_i\left(\sum_{j=1}^{i} x_{i,j}\right),$$

subject to

$$\begin{array}{lll} \sum_{i=j}^{M} x_{i,j} & = \tau'_j - \tau'_{j-1}, & j = 1, \ldots, M, \\ x_{i,j} & \ge 0, & 1 \le j \le i \le M. \end{array}$$

The first constraint equates the length of the jth interval to the amount of service allocated to all of the schedulable tasks in that interval. Note that this implies that the processor is never idle during the interval $[\tau'_0, \tau'_M]$. The second constraint simply states that a task is allotted a nonnegative amount of service in each of its schedulable intervals. Thus the problem is to maximize a concave objective function with M equality constraints and $M(M-1)/2$ inequality constraints. This is a special case of a general resource allocation problem in which the returns from an activity is a concave function of resources allocated to it, and the goal is to obtain an allocation of resources to activities such that the objective function is maximized. Mjelde [31] has discussed several properties of this problem.

The following Kuhn-Tucker conditions [32] are satisfied by any optimal solution to $\boldsymbol{P}$. μ_j denotes the Lagrange multiplier for task j in $\boldsymbol{P}$.

$$g_i(x_i) = \mu_j, \quad 1 \le j \le i \le M, \quad x_{ij} > 0, \tag{1}$$

and

$$g_i(x_i) \le \mu_j, \quad 1 \le j \le i \le M, \quad x_{i,j} = 0 \tag{2}$$

Equation (1) indicates that tasks receiving a nonzero service allocation in an interval j have the same value for the derivative of their reward functions, i.e., have the same marginal reward. Tasks which do not receive service allocation in interval j cannot have a larger value of their reward function derivative as shown in (2), as it should be possible to serve these and provide a lower reward.

Let x_i^* be the amount of service time assigned to task i in any optimal solution. $x_{i,j}^*$ is the service-time allocation to task i in interval j in any optimal solution. Define

$$x^* = \left[x_{1,1}^*, x_{1,2}^*, \ldots, x_{M,M}^*\right]$$

to be an optimal solution to $\boldsymbol{P}$. Let Σ be the set of all optimal solutions to $\boldsymbol{P}$. The following two lemmas are proved in the appendix.

LEMMA 1. *Any* $x^* \in \Sigma$ *satisfies the relations,*

1) *if* $\exists$ *interval* k *s.t.* $x_{i,k}^* > 0$ *and* $x_{j,k}^* > 0$, *then* $g_i(x_i^*) = g_j(x_j^*), 1 \le k \le i, j \le M$.
2) *if* $\exists$ *interval* k *s.t.* $x_{i,k}^* > 0$ *and* $x_{j,k}^* = 0$, *then* $g_i(x_i^*) \ge g_j(x_j^*), 1 \le k \le i, j \le M$.
3) *if* $x_j^* = 0$, *then* $\forall i \in 1, 2, \ldots, (j-1)$ *s.t.* $x_i^* > 0$, $g_i(x_i^*) \ge g_j(x_j^*)$.
4) *if* $\exists$ *task* k s.t. $x_{k,i}^* > 0$ *and* $x_{k,j}^* > 0$, *then* $\mu_i = \mu_j$, $1 \le i, j \le k \le M$.

LEMMA 2. *For every* $x^* \in \Sigma$:

1) *The Lagrange multipliers* μ_i *form a monotonically nonincreasing sequence, i.e.,* $\mu_i \ge \mu_{i+1}$, $1 \le i < M$,
2) x^* *can be constructively transformed into* $x'^* \in \Sigma$, *such that for all of the tasks* i *that receive service in* x^*, *the corresponding* g_i*s in* x'^* *are monotonically nonincreasing, i.e.,* $g_i \ge g_j$, $1 \le i < j \le M$, $x_i^*, x_j^* > 0$.

According to the above lemmas, the optimal solution in any single interval j can be derived by simply choosing the tasks whose reward functions have the highest derivative values and allocating them service till the interval length is consumed. The next theorem shows how we can use this method iteratively to generate an optimal solution to $\boldsymbol{P}$. We introduce an *auxiliary* optimization problem $\boldsymbol{P}_i$, $1 \le i \le M$. Here $\boldsymbol{P}_i$ is a subproblem of $\boldsymbol{P}$ with tasks $i, i+1, \ldots, M$ all having the same arrival time of τ'_{i-1}. Problem $\boldsymbol{P}_1$ is equivalent to problem $\boldsymbol{P}$. The solution algorithm for $\boldsymbol{P}$ sequentially solves problems $\boldsymbol{P}_i$, $i = M, \ldots, 1$. Problem $\boldsymbol{P}_i$ is:

$$\text{maximize } \sum_{k=i}^{M} f_i\left(\sum_{j=i}^{k} x_{k,j}\right),$$

subject to

$$\begin{aligned} &\textstyle\sum_{k=j}^{M} x_{k,j} &&= \tau'_j - \tau'_{j-1}, && j = i, \ldots, M, \\ &x_{k,j} &&\ge 0, && 1 \le j \le k \le M. \end{aligned}$$

The following theorem describes a property satisfied by any optimal solution to problem $\boldsymbol{P}_i$. The top-level algorithms described in the next subsection utilize this property to guarantee an optimal solution to problem $\boldsymbol{P}$. Define $x_j^{*(i)}$ as the service time assigned to task j in the optimal solution for problem $\boldsymbol{P}_i$. Similarly, $\mu_j^{(i)}$ denotes the Langrange multiplier for task j in $\boldsymbol{P}_i$.

THEOREM 1. *Let the optimal solution to problem* $\boldsymbol{P}_{i+1}$ *allocate service time* $x_j^{*(i+1)}$ *to task* j. *Let* $\mathcal{K}$ *be the set of tasks which receive nonzero allocation in the interval* $(\tau'_{i-1}, \tau'_i]$ *in the optimal solution to* $\boldsymbol{P}_i$, *i.e.,* $\mathcal{K} = \{k \mid x_k^{*(i+1)} < x_k^{*(i)}\}$. *Then the optimal solution to* $\boldsymbol{P}_i$ *satisfies:*

$$\sum_{k\in\mathcal{K}} g_k^{-1}\left(\mu^{(i)}\right) = \sum_{k\in\mathcal{K}} x_k^{*(i+1)} + \tau'_i - \tau'_{i-1}$$

where $\mu_k^{(i)} = \mu^{(i)}$ *is the value of the Langrange multiplier for all tasks* $k \in \mathcal{K}$ *in* $\boldsymbol{P}_i$.

PROOF. Since the processor is never idle while tasks have not reached their deadlines,

$$\sum_{k\in\mathcal{K}} \left(x_k^{*(i)} - x_k^{*(i+1)}\right) = \tau'_i - \tau'_{i-1},$$

or

$$\sum_{k\in\mathcal{K}} x_k^{*(i)} = \sum_{k\in\mathcal{K}} x_k^{*(i+1)} + \tau'_i - \tau'_{i-1}. \tag{3}$$

Applying Lemma 1.1 to interval $(t'_{i-1}, t'_i]$ we know that all tasks $k \in \mathcal{K}$ have the same value of g and $\mu^{(i)}$. Lemma 2.2 indicates that the remaining tasks in P_i may have lower g and $\mu^{(i)}$ values. Therefore, (3) is rewritten as:

$$\sum_{k\in\mathcal{K}} g_k^{-1}\left(\mu^{(i)}\right) = \sum_{k\in\mathcal{K}} x_k^{*(i+1)} + \tau'_i - \tau'_{i-1}. \tag{4}$$

□

REMARK. As stated, $\boldsymbol{P}$ assumes that policies are *nonidling*. However, it is a simple exercise to show that, even if this is relaxed, it never pays to idle the processor when the reward functions are nondecreasing.

3.3.2 Top-Level Scheduling Algorithm

This subsection presents the algorithm which computes the optimal service time allocations for task sets. This algorithm uses the results of Theorem 1 to determine the values of x_i that solve problem $\boldsymbol{P}$.

```
1. initialize
      $\mathcal{L} = \emptyset$
      $x_i = 0, \forall i$
2. loop
      for $m = M$ downto 1 do
         Insert $g_m(0)$ into $\mathcal{L}$. Order the tasks in $\mathcal{L}$ in
            decreasing value of $g_i$.
         Let $\pi(i)$ denote the task within $\mathcal{L}$ with the
            $i$th largest $g()$.
         Using a binary search, find $l$ s.t.
            $\sum_{i=1}^{l+1}\left[g_{\pi(i)}^{-1}\left(g_{l+1}\left(x_{l+1}\right)\right) - x_{\pi(i)}\right] =$
               $\tau'_m - \tau'_{m-1} \geq$
               $\sum_{i=1}^{l}\left[g_{\pi(i)}^{-1}\left(g_l\left(x_l\right)\right) - x_{\pi(i)}\right]$      $(\star)$
         Solve for $\mu$ in
            $\sum_{i=1}^{l} g_{\pi(i)}^{-1}(\mu) = \sum_{i=1}^{l} x_{\pi(i)} + \tau'_m - \tau'_{m-1}$
         $x_{\pi(i)} = g_{\pi(i)}^{-1}(\mu), \quad i=1,\ldots,l.$
```

Fig. 2. Algorithm for general reward functions.

The algorithm, given in Fig. 2, solves P_j in the jth iteration in step 2. In this jth iteration, task j becomes schedulable, and hence is added to a list $\mathcal{L}$, in which the elements are ordered in decreasing values of their derivatives. Then a binary search is performed on this list of schedulable tasks, computing how many of these tasks can actually be scheduled using the condition outlined in the theorem. The search finds a number l such that the l tasks with the highest reward rates will be allocated additional service from $(\tau'_{j-1}, \tau'_j]$. The value of μ is then computed using (4) from which the values of the $x_i^{(j)}$s are obtained.

In this version, computing each sum during the binary search takes time $O(M)$ and there are $O(\log M)$ probes done in the binary search. Thus each iteration takes time $O(M \log M)$ and the running time of the algorithm is $O(M^2 \log M)$.

This algorithm can be improved by grouping tasks having the same derivatives into a single set. This improved algorithm then maintains values of the summands in line $(\star)$ of Fig. 2 for each of the sets and otherwise executes the previous algorithm on the sets. The ability to maintain precomputed values of the sum permits a reduction in the running time of the algorithm to $O(M^2)$. When tasks have exponential reward functions, the scheduling complexity reduces further to $O(M \log M)$. Complete descriptions of these algorithms can be found in [33].

3.1.3 The Lower-Level Policy

The purpose of the lower-level algorithm is to actually schedule tasks using the service time allocation information computed by the top-level policy. In the absence of future arrivals, it executes every task for its allocated time. If an arrival occurs, then the top-level policy is executed again. There are many ways that the tasks can be scheduled once their service allocations are known. In the presence of arrivals, different policies will perform differently. In this section, we study two representative lower-level policies, earliest deadline first (EDF) and first come first serve (FCFS), and compare their performance.

The rationale for choosing EDF and FCFS is that they are simple, and generate very few preemptions per task. A potential disadvantage is that in the presence of task arrivals, these policies favor tasks with earlier deadlines and arrival times respectively, and thus the marginal reward rates of tasks may not be balanced as in (1) at the time of a new task arrival.

Intuitively, we would not expect FCFS to perform as well as EDF for the following reason. The allocations of the top-level policy satisfy the first constraint of the optimization problem P. Therefore, for a static set of tasks, it is trivial to show that EDF is guaranteed to accord each task the service time allocated to it by the top-level policy [34]. FCFS, however, cannot provide that guarantee. Fig. 3 shows an example where FCFS is unable to provide the top-level policy's allocation to every task. There are two tasks, numbered 1 and 2, with $[a_1, \tau_1] = [0, 3]$, $[a_2, \tau_2] = [1, 2]$, as shown in Fig. 3.i. Fig. 3.ii depicts the linear reward functions of the two tasks with reward rates $g_1 = 1$, $g_2 = 2$. Fig. 3.iii shows the top-level policy's allocation decisions made at times $t = 0$ and $t = 1$. At time $t = 0$, task 1 receives all three units of allocation, and at time $t = 1$, both tasks receive one unit allocation. But FCFS schedules task 1 in the time-interval (1, 2], as shown in Fig. 3.iv which causes task 2 to leave the system without receiving any service. The processor is left idle in time-interval (2, 3]. The schedule under EDF is shown in Fig. 3.v as a comparison.

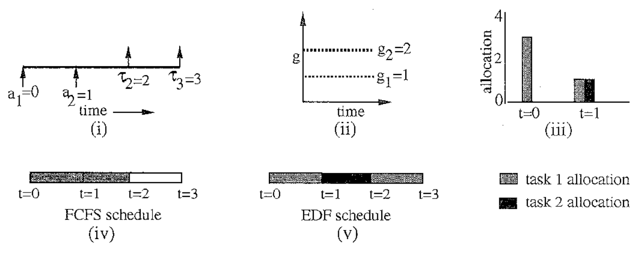


Fig. 3. FCFS vs. EDF scheduling.

3.2 Greedy Policy

The intuition behind the top-level policy is to allocate service times to tasks that return high reward rates. Equation (1), the condition for optimality, and Theorem 1 indicate that the incremental reward rates of the serviced tasks are to be balanced subject to the constraint of task deadlines. Thus, we consider a policy that executes tasks in such a way as to keep the marginal reward rates as equal as possible. This policy is greedy because it maximizes the instantaneous reward rate by prioritizing tasks on the basis of their marginal rewards. This policy, a simple variation of the processor sharing policy, is called the balanced reward processor sharing policy (BRPS). Although it is an idealized and unimplementable policy, can be approximated by a round robin policy. Its drawback is that it would be characterized by a large number of task preemptions. Note that instead of doing a global optimization as is done by the top-level algorithm when scheduling a set of tasks with identical arrival times, this greedy policy achieves a local optima in every interval $(\tau'_{j-1}, \tau'_j]$ in Fig. 1. When arrivals are frequent, this greedy approach of reward maximization could yield higher reward rates than using FCFS or EDF in the two-level policy, for reasons outlined in Subsection 3.1.3. On the other hand, it can be easily shown that, for a set of tasks with identical arrival times, combining local optimum solutions in this manner does not lead to a globally optimum solution. Section 5 contrasts the benefit of doing global optimization assuming no future arrivals and then scheduling tasks using a simple lower-level policy, versus scheduling to achieve local optimas using this greedy method.

4 Analytical Model and Upper Bounds

In this section, we develop an analytical model of an IRIS task system and obtain upper bounds on the reward rate that is achievable by any scheduling policy in this system.

The task system is modeled as a single server system servicing $K > 0$ classes of tasks, labeled $k = 1, \ldots, K$. Tasks from class k arrive to the system at arbitrary times according to a stationary process with rate λ_k. We further assume that the laxities for the different task classes form mutually independent sequences of independent and identically distributed (i.i.d.) random variables (r.v.s) with means $E[\tau_k]$, $k = 1, 2, \ldots, K$. Associated with a class k task is a nondecreasing concave reward function $f_k : \mathbb{R}_+ \to \mathbb{R}$, and a task of this class accrues a reward of $f_k(x)$ upon receiving x units of service. Observe that the reward function depends on the task class but not on the identity of the task.

Denote the service time accrued by task i of class k under a policy π by $A^{\pi}_{k,i}$ and the associated reward by $R^{\pi}_{k,i} = f_k(A^{\pi}_{k,i})$ with means $E[A^{\pi}_k]$ and $E[R^{\pi}_k]$, respectively. Denote the service time accrued by a randomly chosen task and a randomly chosen class k task by A and A_k, respectively. We only consider policies such that $R^{\pi}_k = \lim_{i\to\infty} R^{\pi}_{k,i}$ exists $\forall k$. We are interested in the average reward per task, $E[R^{\pi}]$, and the average accrued reward per unit time, $\gamma^{\pi} = \lambda E[R^{\pi}]$, where $\lambda = \sum_{k=1}^{K} \lambda_k$.

We now derive two upper bounds on the reward rate for any such policy. The first upper bound makes no assumption regarding the arrival process while the second upper bound requires that the tasks arrivals be described by Poisson processes. The latter upper bound is tighter than the first.

The average reward per task of any scheduling policy is defined as:

$$\begin{aligned} E[R^{\pi}] &= \sum_{k=1}^{K} \frac{\lambda_k}{\lambda} E[R^{\pi}_k], \\ &= \sum_{k=1}^{K} \frac{\lambda_k}{\lambda} E[f_k(A^{\pi}_k)], \\ &\le \sum_{k=1}^{K} \frac{\lambda_k}{\lambda} f_k(E[A^{\pi}_k]), \end{aligned}$$

using Jensen's inequality [35].

Thus the average reward per unit time for any scheduling policy is bounded by:

$$\begin{aligned} \gamma^{\pi} &\le \sum_{k=1}^{K} \lambda_k f_k(E[A^{\pi}_k]), \\ &\le Q^{\pi}, \end{aligned} \tag{5}$$

where Q^{π} is the solution of the following non-linear optimization problem:

$$\text{maximize} \quad \sum_{k=1}^{K} \lambda_k f_k(y_k),$$

subject to

$$\begin{aligned} \textstyle\sum_{k=1}^{K} \lambda_k y_k &= C, \quad j = 1, \ldots, M, \\ y_k &\ge 0, \quad 1 \le k \le K, \end{aligned}$$

for any $C \ge \sum_{k=1}^{K} \lambda_k E[A_k]$.

When the arrival processes of each task class is completely general, the system can be modeled as a $G/G/\infty$ queue. This is due to the fact that every task departs from the system at its deadline, irrespective of the number of other tasks that are present in the system at its arrival time. In this case we are unable to obtain an exact expression for $\sum_{k=1}^{K} \lambda_k E[A_k]$. However, we have the following bound:

$$\sum_{k=1}^{K} \lambda_k E[A_k] \le \begin{cases} 1, & 1/\lambda < 1/\lambda\left(\sum_{k=1}^{K} \lambda_k E[\tau_k]\right), \\ \left(\sum_{k=1}^{K} \lambda_k E[\tau_k]\right), & \text{otherwise.} \end{cases} \tag{6}$$

The first term corresponds to the server never being empty, whereas the second term corresponds to a system in which there is never any overlap of different tasks in the system.

When the arrival process is Poisson, we can derive an exact expression for $\sum_{k=1}^{K} \lambda_k E[A_k]$. Let B denote the length of a $M/G/\infty$ busy period. Let $E[N^{(b)}]$ and $E[N^{(b)}_k]$ denote the expected number of tasks and the expected number of tasks of class k served in a busy period, $E[N^{(b)}_k] = \frac{\lambda_k}{\lambda} E[N^{(b)}]$, $k = 1, \ldots, K$. The number of tasks in the system, N, is a Poisson r.v. with $\Pr[N = n] = \rho^n e^{-\rho}/n!$, $n = 0, 1, \ldots$, where $\rho = \sum_{k=1}^{K} \lambda_k E[\tau_k]$. Therefore:

$$\sum_{k=1}^{K} E[N^{(b)}_k] E[A_k] = E[B]. \tag{7}$$

The mean number of tasks served in a busy period is given by $E[N^{(b)}] = \lambda(E[B] + 1/\lambda)$ which, when substituted into (7) yields

$$\sum_{k=1}^{K} \lambda_k E[A_k] = \frac{E[B]}{E[B] + 1/\lambda}.$$

Now, $\Pr[N = 0]$ can also be expressed in terms of $E[B]$ as $\mathrm{pr}[N = 0] = \frac{1/\lambda}{E[B]+1/\lambda}$ which yields $E[B] = (e^{\rho} - 1)/\lambda$. Thus,

$$\sum_{k=1}^{K} \lambda_k E[A_k] = 1 - e^{-\rho}. \qquad (8)$$

The two optimization problems are solved using algorithms similar to the one presented for problem *P*.

5 Numerical Results

In this section, we examine the performance of our scheduling policies. We have used two metrics to quantify the performance of our policies:

- The reward rates generated under a policy. This is the basic metric to evaluate a policy that schedules IRIS tasks.
- Task preemptions in a processor leads to context switch overheads which reduce the available processor time to execute tasks. Hence the second metric that we have used is the average number of times that a task is preempted by another task under a policy.

To evaluate the policies, we simulate a uniprocessor system with two classes of tasks arriving to it. The reward function for a class k task is of the form $f_k(x) = 1 - e^{-\delta_k x}$, k = 1, 2. The motivation for using exponential reward functions come from Dean and Boddy [4], [2], who have employed an IRIS algorithm for time-dependent planning problems, that has an exponential reward function. Horvitz [16] also has described IRIS computations with exponential reward functions.

The independent parameter we have used for our simulations is the processor utilization U, a measure of the processor load, due to the combination of two task classes. Using this as the independent parameter allows us to examine the performance of our scheduling policies under various processor loads.

The arrival process and laxity distribution of class k are Poisson with rate λ_k and exponential with parameter τ_k, respectively. We define $\rho_k = \lambda_k E[\tau_k]$. From Section 4, we know that the probability of there being no tasks present in this system is $P[N = 0] = e^{-\rho}$. Hence, the processor utilization, or the fraction of the time that the processor is busy, is given by $U = 1 - e^{-\rho}$. Analogously, $1 - e^{-\rho_k}$ represents the load on the processor owing to class k tasks.

We have performed simulations with 48 different parameter sets, by varying task arrival rates, laxity parameters and reward rates. The simulations have been performed by varying ratios of ρ_1, ρ_2, (ρ_1/ρ_2 = 0.5, 1, 2, 4) and ratios of δ_1, δ_2, (δ_1/δ_2 = 2, 3, 5, 10). Class 1 thus corresponds to the class with the higher reward rate. Varying the ratio ρ_1/ρ_2 allows us to examine the performance of the policies at various load mixes of higher reward rate and lower reward rate tasks. For each value of ρ_1/ρ_2, we have varied the ratio of $E[\tau_1]$ and $E[\tau_2]$ ($E[\tau_2]/E[\tau_1]$ = 1, 2, 4). Note that $\lambda_1/\lambda_2 = \rho_1 E[\tau_2]/\rho_2 E[\tau_1]$. In all simulations, the numerical values assigned to δ_2 and $E[\tau_2]$ are 0.4 and 10, respectively. The values of λ_1 and λ_2 are computed using the value of U. Each point estimate is the average of 19 replications, each consisting of 50,000 task completions. The 95% confidence interval half-width are less than 1 percent of the point value in all the simulation results. We present a few representative results of the 48 simulations for each of the metrics.

TABLE 1
Comparison Between Performance of EDF, BRPS, and FCFS Policies

U	γ^E	γ^B	γ^F	γ_1^E	γ_1^B	γ_1^F
0.05	0.0044	0.0044	0.0044	0.0016	0.0016	0.0016
0.10	0.0089	0.0089	0.0089	0.0033	0.0033	0.0033
0.20	0.0189	0.0188	0.0187	0.0071	0.0071	0.0067
0.30	0.0301	0.0299	0.0292	0.0113	0.0113	0.0109
0.40	0.0430	0.0426	0.0407	0.0162	0.0161	0.0152
0.50	0.0581	0.0573	0.0532	0.0219	0.0218	0.0199
0.60	0.0764	0.0751	0.0667	0.0289	0.0288	0.0251
0.70	0.0994	0.0973	0.0817	0.0379	0.0377	0.0307
0.80	0.1307	0.1275	0.0987	0.0505	0.0501	0.0373
0.90	0.1806	0.1758	0.1198	0.0717	0.0710	0.0457
0.95	0.2253	0.2197	0.1342	0.0923	0.0914	0.0518

Parameters: $\lambda_1/\lambda_2 = 0.5$, $\tau_2/\tau_1 = 1$, $\rho_1/\rho_2 = 0.5$, $\delta_1/\delta_2 = 5$

U	γ^E	γ^B	γ^F	γ_1^E	γ_1^B	γ_1^F
0.05	0.0172	0.0172	0.0172	0.0163	0.0162	0.0162
0.10	0.0354	0.0352	0.0351	0.0334	0.0333	0.0331
0.20	0.0746	0.0742	0.0725	0.0705	0.0700	0.0684
0.30	0.1188	0.1176	0.1118	0.1122	0.1111	0.1054
0.40	0.1693	0.1669	0.1527	0.1600	0.1577	0.1436
0.50	0.2280	0.2238	0.1947	0.2157	0.2117	0.1828
0.60	0.2981	0.2914	0.2374	0.2825	0.2760	0.2223
0.70	0.3852	0.3751	0.2802	0.3659	0.3558	0.2615
0.80	0.5004	0.4860	0.3225	0.4775	0.4625	0.2992
0.90	0.6746	0.6560	0.3619	0.6494	0.6282	0.3319
0.95	0.8207	0.8020	0.3785	0.7969	0.7739	0.3431

Parameters: $\lambda_1/\lambda_2 = 16$, $\tau_2/\tau_1 = 4$, $\rho_1/\rho_2 = 4$, $\delta_1/\delta_2 = 5$

The first set of results, tabulated in Table 1, illustrates, the difference between using EDF and FCFS in conjunction with the top-level policy and the BRPS policy. Although we have tabulated and analyzed the results of only two simulations, we emphasize that all the other simulation results yield similar conclusions. For these simulations, the processor utilization U varies between 0.5 and 0.95. The tables list the average reward rates attained by scheduling both classes of tasks, and the reward rate attained by the class 1 tasks. The reward rate of class 2 tasks is the difference between the cumulative reward rate and the class 1 reward rate. The first three columns, γ^E, γ^B, and γ^F, tabulate the reward rates attained by using EDF, BRPS, and FCFS policies. The next three columns tabulate the corresponding class 1 reward rates. For very low processor utilizations (U < 0.4), the reward rates attained by all three policies are the same. In this region, the task arrival rate is very low and a typical busy period consists of only one task. Hence each task gets executed till their deadline under any policy. When U is higher, using EDF as the lower-level policy in the two-level scheduling approach provides the best reward rate. The performance of the BRPS policy comes very close, with γ^B always remaining within 97% of γ^E. As expected, the FCFS policy yields much smaller reward rates. Analyzing the class 1 reward rates leads to the same conclusions. Taking into consideration that processor sharing is an idealized abstraction of the traditional round-robin algorithm under which the number of preemptions is a much higher than under EDF, we prefer the combination of the top-level algorithm and EDF over BRPS. Note that the number of preemptions under FCFS is also very small, and the primary reason for lower reward rates under FCFS is the idling of the processor as shown in Fig. 3, even when a task has not reached its deadline. This can be ameliorated by modifications to the two-level policy so that the top-level policy is re-executed when such a situation occurs. The modifications will result in a policy that generates better reward rates, but at the cost of additional scheduling complexity. Considering that the reward rates achieved using EDF in the two-level policy is extremely close to the upper bounds as shown in the next paragraph, we regard EDF as the lower-level algorithm of choice. For all the other graphs in this section, the performance numbers reported are for EDF.

Fig. 4 compares the upper bounds and the reward rates achieved by our two-level scheduling policy with EDF as the lower-level algorithm. We analyze three graphs with $\rho_1/\rho_2 = 0.5$, 1, and 4, representing systems where the class 1 load is less than, equal to, and greater than the class 2 load on the processor. In all the graphs, $\delta_1/\delta_2 = 5$. The processor utilization, U, is varied between 0.5 and 0.998. γ_G^{ub} and γ_M^{ub} are the upper bound curves for tasks with general arrival patterns and Poisson arrivals, respectively. γ, γ_1, and γ_2 represent the cumulative reward rate, the class 1 reward rate and the class 2 reward rate. There are several interesting aspects to this figure. First, although γ_M^{ub} is tighter than γ_G^{ub} as noted in Section 4, the difference between them is marginal. In fact, the two curves are virtually identical for U less that 0.1 and greater than 0.95. As $U \to 1$, (6) and (8), the constraints of the two upper bound optimization problems in the previous section, become identical. When $U \to 0$, then $\lambda \to 0$, which in turn implies Q^{π} in (5) tends to zero. This explains the convergence of the two curves at their endpoints.

The second observation is that the cumulative reward rates are very close to the upper bounds in all the simulations. Again the γ curves come closest to the upper bounds for very low and high values of U. As the processor utilization increases ($U \to 1$), $\lambda \to \infty$, i.e., asymptotically the interarrival time goes down to zero. When the interarrival time is infinitesimally small, there are always enough class 1 tasks so that no class 2 tasks receive any service. A class 1 task receives only an infinitesimally small amount of service before being preempted by the arrival of another class 1 task. Therefore γ_G^{ub}, γ_M^{ub}, γ, and γ_1, all converge to δ_1.

The third aspect of Fig. 4 is the behavior of the class-wise reward rates, γ_1 and γ_2. The class-wise reward rate characteristics change as ρ_1/ρ_2 increases. In the first graph, where the arrival rate of class 1 is lower than that of class 2, we notice that, although γ_1 is initially lower than γ_2, it increases rapidly with processor utilization and catches up with γ_2. In the other two graphs, where λ_1 is the same and greater than λ_2, respectively, we note that γ_1 is consistently higher than γ_2 and is very close to γ. As ρ_1/ρ_2 increases, i.e., the class 1 task load on the processor increases, γ_1 approaches γ whereas γ_2 approaches zero. The same trend is also observed as δ_1/δ_2 is increased (not shown here). In the last two graphs, asymptotically γ_1 equals γ, while γ_2 is asymptotically zero. This asymptotic behavior is explained in the previous paragraph.

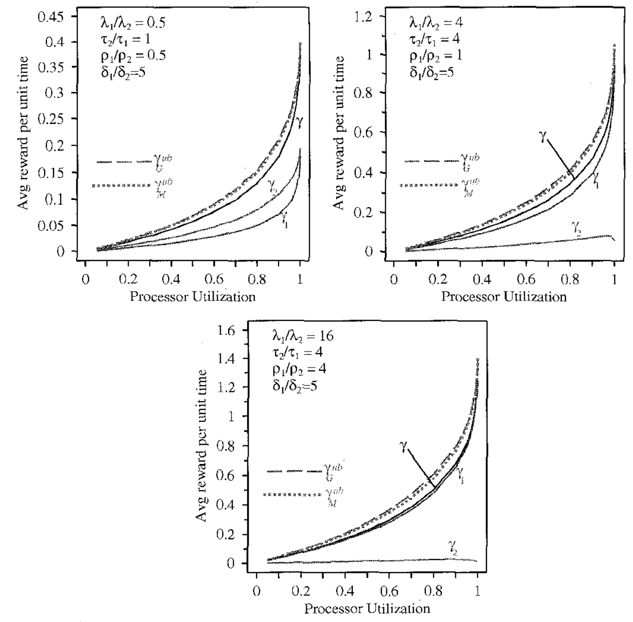


Fig. 4. Average reward attained.

Fig. 5 illustrates γ_1^{iso}/γ_1 and γ_2^{iso}/γ_2, the ratios of the reward rates when the task classes execute in isolation (γ_1^{iso} and γ_2^{iso}) to the reward rates when they execute in presence of each other. The data plotted are for the same set of parameters values as Fig. 4. The graphs describe the effect of an additional task class on the reward rate of a class under our scheduling policy. These ratios are indicative of the degree of fairness of the policy by depicting the effect of preference of the policy for one class over another. When $\lambda_1 < \lambda_2$, both γ_1^{iso}/γ_1 and γ_2^{iso}/γ_2 are almost the same and equal to 1 for $U < 0.45$, i.e., the class-wise reward rates are not affected by the presence of the other class. As the processor load increases, the effect of the presence of another task class shows on both the classes with the ratios increasing marginally above 1. As expected, the effect on class 2 is slightly more marked than on class 1. When $\lambda_1 \geq \lambda_2$, the presence of class 2 tasks has no effect on the reward rate of class 1 tasks. However class 2 tasks pay a toll as the maximum value of γ_2^{iso}/γ_2 sharply increases to 3.7 and 12.5 in the latter two graphs. This is a result of the top-level scheduling policy striving to maximize the cumulative reward rate, at the cost of class 2 tasks. Since $\gamma_2 \to 0$ as $U \to 1$ for the latter two parameter sets, the γ_2^{iso}/γ_2 ratio asymptotically goes to infinity. A real-time system is often engineered to operate at moderate loads. For such loads, the scheduling policy is quite fair to both task classes and the reward rates they achieve are almost the same as they would achieve when executing in isolation.

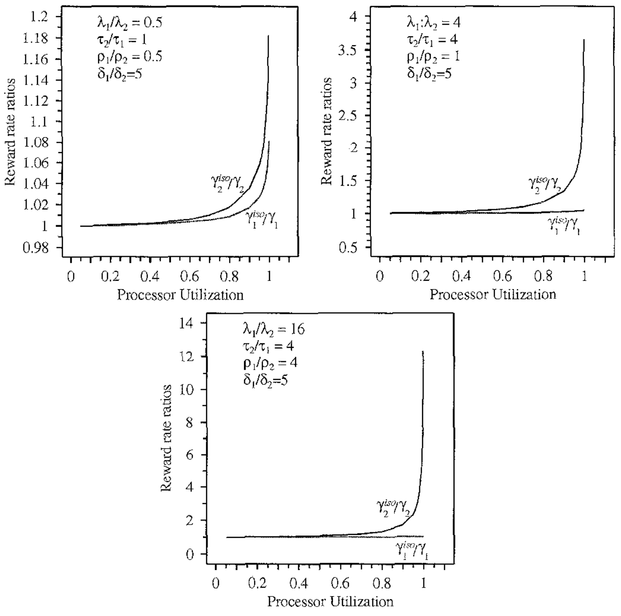


Fig. 5. Reward loss in presence of other task class.

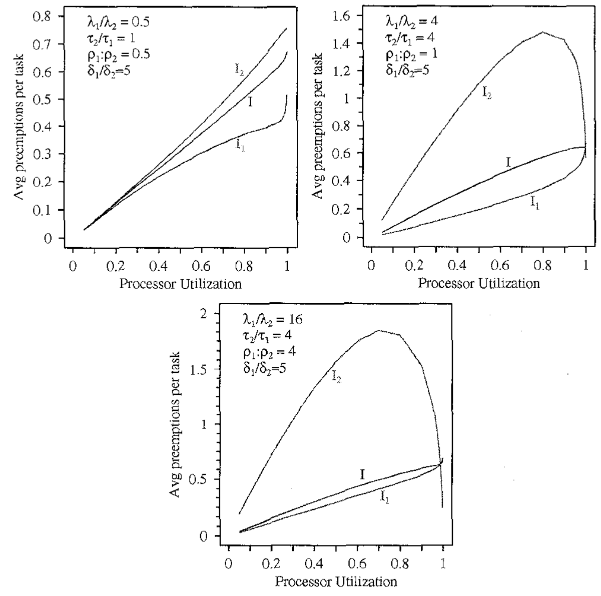


Fig. 6. Expected number of preemptions per task.

The second metric used to evaluate the quality of our scheduling policy is the average number of preemptions seen by a task. Fig. 6 graphs I, I_1, and I_2 which denote the cumulative average, the class 1 average and the class 2 average of the number of preemptions respectively, when EDF is used as the lower-level policy. As expected, I_2 is always higher than I_1, and the cumulative average I lies in between them. We note that I and I_1 are always less than 1, and I_2 is less than 2 in all the graphs. Hence, in general, using EDF as the lower-level policy results in a low number of preemptions. In all three graphs, the slope of I_2 is greater than that of I_1 when processor utilizations are less than 0.8. As the arrival rates increase, class 2 tasks are interrupted more frequently by class 1 tasks and newly arrived class 2 tasks, whereas class 1 tasks are only interrupted by other class 1 tasks. Hence the slope of I_2 is higher than I_1 in this range. In contrast to this behavior, for U in the range [0.8, 1], I_1 goes up while I_2 drops down sharply towards zero. This apparent anomaly can again be explained by the proliferation of class 1 tasks resulting in class 2 tasks not being scheduled at all. Thus, a high fraction of the preemptions are caused by a newly arrived class 1 task preempting another class 1 tasks. In this region, therefore, I equals I_1. Overall, when real-time systems operate in the low processor utilization region, the average number of preemptions seen by tasks is likely to be quite low.

6 Extensions

In this section, we discuss two simple extensions. The first is an extension of the scheduling algorithm for IRIS tasks which have piecewise-linear reward functions. The second is a modification of the top-level scheduling algorithm for handling IRIS tasks that need to receive a certain minimum execution time.

6.1 Algorithm for Piecewise-Linear Reward Functions

Piecewise-linear reward functions are useful in that arbitrary non-linear functions can be approximated by piecewise-linear functions. The piecewise linear function for task k with n_k segments is represented by two vectors: $(g_k(1), g_k(2), \ldots, g_k(n_k))$, and $(l_k(0), l_k(1), \ldots, l_k(n_k))$. $g_k(i)$ is the slope of segment i, while $l_k(i)$ is the length of the projection of segment i on the x-axis. A piecewise linear function with three segments is shown in Fig. 7. Thus function $f_k(x)$ is defined as:

$$f_k(x) = \sum_{i=1}^{j} g_k(i) l_k(i) + g_k(j+1)\left(x - \sum_{i=1}^{j} l_k(i)\right),$$

where $\sum_{i=1}^{j} l_k(i) < x \le \sum_{i=1}^{j+1} l_k(i)$.

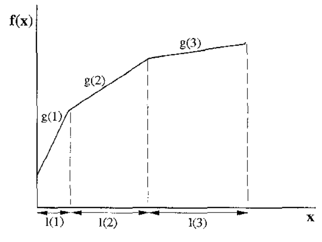


Fig. 7. A typical piecewise linear reward function.

For any segment s, we also define the following: $task(s) \equiv k$ if s is the ith segment defining f_k, $1 \le i \le n_k$, and $len(s) \equiv l_{task(s)}(i)$. Thus $task(s)$ represents the task whose reward function contains segment s, and $len(s)$ represents the length of the projection of the segment on the x-axis.

The algorithm is described in Fig. 8. When task m becomes schedulable in step m, its first segment is inserted into a list $\mathcal{L}$ using a binary search. After that, the segment with the largest derivative in $\mathcal{L}$ is selected and allocated service time from the interval. When the service allocation to any segment s equals *len*(s), it is not pushed back into $\mathcal{L}$. Instead the next segment from *task*(s) is inserted into $\mathcal{L}$. To compute the complexity of this algorithm, we note that the length of $\mathcal{L}$ is at most M. Any segment of a reward function for a task is inserted into the list $\mathcal{L}$ at most once and then accessed at most M times, once for every subinterval. Thus the complexity of this algorithm is $O(\log M \sum_{k=1}^{M} n_k)$ when there are M tasks and the reward function of task k consists of n_k pieces.

```
1. initialize
   $\mathcal{L}$ = all the segments of all the tasks in decreasing
     order of segment derivatives
   $x_k = 0 \ \forall k = 1, 2, \ldots, M$
2. loop
   for $m = M$ downto 1 do
     Using a binary search, insert the first segment
       of task $m$ into $\mathcal{L}$
     $time\_allocation\_left = \tau'_m - \tau'_{m-1}$
     while ($time\_allocation\_left > 0$) do
       /* Allocate time to the segment with */
       /* the largest derivative */
       Get next segment $s \in \mathcal{L}$
       if ($time\_allocation\_left \geq len(s)$)
         /* Allocation for corresponding task */
         /* can increase by $len(s)$ */
         $time\_allocation\_left =$
           $time\_allocation\_left - len(s)$
         $x_{task(s)} = x_{task(s)} + len(s)$
         Using a binary search, insert the next
           segment of $task(s)$ into $\mathcal{L}$
       else
         /* All of $time\_allocation\_left$ */
         /* is consumed */
         $len(s) = len(s) - time\_allocation\_left$
         $x_{task(s)} = x_{task(s)} + time\_allocation\_left$
         $time\_allocation\_left = 0$
         Push $s$ to front of $\mathcal{L}$
```

Fig. 8. Algorithm for piecewise-linear reward functions.

6.2 IRIS Tasks with Mandatory Portions

In the IRIS model there is no notion of a mandatory minimum service time that a task must receive. However, the top-level algorithms can easily handle, with a slight modification, tasks having such constraints. Fig. 9 describes the algorithm which transforms the static problem into one where the deadlines of the tasks have been reduced to account for the mandatory portions. In the algorithm, m_i denotes the length of the mandatory phase of task i. The rest of the notation is defined in Section 3.1.1.

When mandatory subtasks are introduced, the IRIS notion of tasks being able to receive any amount of service before their deadlines does not hold. Hence there is an additional performance metric, the probability of missing a task's deadline before it can receive its mandatory amount of service. While scheduling tasks in the *imprecise computation model* [25], it is assumed that mandatory portions of tasks can always be given service before their deadlines. Further discussion of this is beyond the scope of this paper.

```
1. initialize
   $\tau^*_M = \tau'_M - m_M$
2. loop
   for $i = M - 1$ downto 1 do
     $\tau^*_i = \min\{\tau^*_{i+1}, \tau'_i\} - m_i$
3. initialize
   $total\_m = m_1$
4. loop
   for $i = 2$ to $M$ do
     $\tau^*_i = \tau^*_i - total\_m$
     $total\_m = total\_m + m_i$
5. loop
   for $i = 1$ to $M$ do
     $f^*_i(x) = f_i(x + m_i)$
```

Fig. 9. Transforming tasks with mandatory phases into tasks without them.

7 Conclusion

In this paper, we have presented the IRIS model, a new real-time task model. These real-time tasks represent computations whose quality increases with execution time. To characterize this computation, a reward function is associated with every task indicating the reward generated by the task as a function of execution time. We have developed a two-level scheduling policy for IRIS tasks executing on a uniprocessor system. We have presented an algorithm for the static version of the problem which schedules tasks to maximize the total reward attained by the task set and used the algorithm as a top-level heuristic in our two-level policy. We have also described a greedy policy and compared its performance to the two-level policy. We have computed theoretical upper bounds on the performance on any policy scheduling IRIS tasks. We have found that with an appropriate choice of a lower-level policy, the performance of our two-level policy is extremely close to the upper bounds. The average number of preemptions of a task under this policy is also very small (less than 2) when EDF is used as the lower-level policy.

Future work on this problem will include extending our approach to scheduling IRIS tasks to a multiprocessor setting, as well as studying the advantages that might accrue by introducing a coprocessor to execute the scheduling algorithms.

Appendix

In this section, we present the proofs of Lemmas 1 and 2.

Lemma 1 Proof.

1) This property is obtained by applying (1) to tasks i and j in interval k.

2) This property is obtained by using (2) with task j in interval k, and using (1) with task i in interval k.
3) Since $x_i^* > 0$, $\exists$ some interval $k \le i$, s.t. $x_{i,k}^* > 0$. Since $i < j$, $k < j$. Thus the proof follows from property 2 of this lemma.
4) This property is obtained by using (1) for intervals i and j with task k. □

LEMMA 2 PROOF.

1) Assume the lemma is not true, i.e., $\mu_i < \mu_{i+1}$ for some i. From Lemma 1.4, we know that there does not exist any task which is scheduled in both intervals i and $(i + 1)$. Therefore there must be some task, m, in the optimal solution which is scheduled in interval $(i + 1)$ but not in interval i. Then from (2) and (1), we get

$$g_m\left(x_m^*\right) \le \mu_i, \text{ and } g_m\left(x_m^*\right) = \mu_{i+1}.$$

This implies $\mu_{i+1} \le \mu_i$, which is a contradiction.

2) We construct x'^* from x^* by an iterative procedure which operates on every pair of tasks which have received nonzero service in x^*.

Take two tasks i and j, $i < j$, that have received nonzero service in x^*. Let $m = \max\{k \mid x_{i,k} > 0\}$, and $n = \max\{k \mid x_{j,k} > 0\}$. The construction is divided into three cases.

Case 1. $M = n$. Then from Lemma 1.1, we know that $g_i(x_i^*) = g_j(x_j^*)$. Construct $x'^*_{i,k} = x^*_{i,k}$ and $x'^*_{j,k} = x^*_{j,k}$, $\forall 1 \le k \le M$.

Case 2. $M < n$. Then we get $g_i(x_i^*) \ge g_j(x_j^*)$ by using (1) and Lemma 2.1. Again construct x'^*_i and x'^*_j as in Case 1.

Case 3. $M > n$. Since $\tau'_j > \tau'_i$, we can construct x'^* that differ from x^* only in assignments to tasks i and j in the mth and nth intervals as follows:

$$x'^*_{k,l} = x^*_{k,l}, \quad k \ne i, j, \forall l = l, \ldots, M,$$

$$x'^*_{i,m} = x^*_{i,m} - \min\left(x^*_{i,m}, x^*_{j,n}\right),$$

$$x'^*_{j,m} = x^*_{j,m} + \min\left(x^*_{i,m}, x^*_{j,n}\right),$$

$$x'^*_{i,n} = x^*_{i,n} + \min\left(x^*_{i,m}, x^*_{j,n}\right),$$

$$x'^*_{j,n} = x^*_{j,n} - \min\left(x^*_{i,m}, x^*_{j,n}\right).$$

Now, if

$$x'^*_{i,m} > 0, \ \max\{k \mid x'^*_{i,k} > 0\} = \max\{k \mid x'^*_{j,k} > 0\} = m,$$

and case 1 can now be applied. If

$$x'^*_{i,m} = 0, \ \max\{k \mid x'^*_{i,k} > 0\} < \max\{k \mid x'^*_{j,k} > 0\} = m,$$

and case 2 can now be applied.

By repeatedly applying this construction procedure to all pairs of tasks that received nonzero service in x^*, we obtain x'^*, in which case 1 or case 2 apply to all pairs. □

ACKNOWLEDGMENTS

The work reported in this paper was supported by ARPA under ESD/AVS contract number F19628-92-C-0089d and by the National Science Foundation under grants NCR-9116183 and CCR-9119922.

REFERENCES

[1] J.A. Stankovic and K. Ramamritham, *Hard Real-Time Systems Tutorial Text*. IEEE Press, 1988.

[2] M. Boddy and T. Dean, "Solving Time-Dependent Planning Problems," *Proc. 11th Int'l Joint Conf. Artificial Intelligence (IJCAI-89)*, pp. 979–984, Detroit, Aug. 1989.

[3] M. Boddy and T. Dean, "Deliberation Scheduling for Problem Solving in Time-Constrained Environments," *Artificial Intelligence*, vol. 67, no. 2, pp. 245–285, June 1994.

[4] T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," *Proc. Seventh Nat'l Conf. Artificial Intelligence (AAAI-88)*, pp. 49–54, St. Paul, Minn., Aug. 1988.

[5] K.S. Decker, V.R. Lesser, and R.C. Whitehair, "Extending a Blackboard Architecture for Approximate Processing," *J. Real-Time Systems*, vol. 2, pp. 47–79, 1990.

[6] J. Liu, "Timing Constraints and Algorithms," *Report on the Embedded AI Languages Workshop*, pp. 9–11, Univ. of Michigan, Nov. 1988.

[7] E. Chang and A. Zakhor, "Scalable Video Coding Using 3-D Subband Velocity Coding and Multi-Rate Quantization," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing*, Minneapolis, July 1993.

[8] E. Chang and A. Zakhor, "Scalable Video Data Placement on Parallel Disk Arrays," *Proc. IS&T/SPIE Symp. Electronic Imaging Science and Technology*, pp. 208–221, San Jose, Calif., Feb. 1994.

[9] International Organization for Standardization, "ISO/IEC JTC1/SC29/WG11 MPEG93/457, Coding of Moving Pictures and Associated Audio, MPEG-2 Systems Working Draft," Nov. 1993.

[10] C.J. Turner and L.L. Peterson, "Image Transfer: An End-to-End Design," *Proc. SIGCOMM Symp. Comm. Architectures and Protocols*, pp. 258–268, Baltimore, Aug. 1992.

[11] B. Hayes-Roth, "Architectural Foundations for Real-Time Performance in Intelligent Agents," *J. Real-Time Systems*, vol. 2, pp. 99–125, 1990.

[12] R.E. Korf, "Depth-Limited Search for Real-Time Problem Solving," *J. Real-Time Systems*, vol. 2, pp. 7–24, 1990.

[13] R.E. Korf, "Real-Time Heuristic Search," *Artificial Intelligence*, vol. 42, no. 2, pp. 189–212, 1990.

[14] K.P. Smith and J.W.S. Liu, "Monotonically Improving Approximate Answers to Relational Algebra Queries," *Proc. Compsac*, Sept. 1989.

[15] S.V. Vrbsky and J.W.S. Liu, "Producing Monotonically Improving Approximate Answers to Database Queries," *Proc. IEEE Workshop Imprecise and Approximate Computation*, pp. 72–76, Phoenix, Ariz., Dec. 1992.

[16] E.J. Horvitz, "Reasoning under Varying and Uncertain Resource Constraints," *Proc. Seventh Nat'l Conf. Artificial Intelligence (AAAI-88)*, pp. 111–116, St. Paul, Minn., Aug. 1988.

[17] S. Zilberstein, "Operational Rationality through Compilation of Anytime Algorithms," PhD thesis, Univ. of California at Berkeley, 1993.

[18] E.G. Coffman, A. Lenstra, and Z. Liu, *Scheduling Theory and Its Applications*. New York: Wiley, 1995.

[19] E.G. Coffman et al., *Computer and Job-Shop Scheduling Theory*. New York: Wiley, 1976.

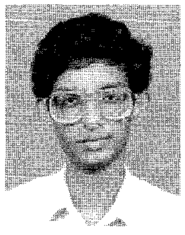
[20] A. Van Tilborg, *Foundations of Real-Time Computing: Scheduling and Resource Management*. Kluwer, 1991.

[21] J.A. Stankovic et al., "Implications of Classical Scheduling Results for Real-time Systems," *Computer*, 1995.

[22] D. Towsley, "Applications of Sample Path Analysis Techniques to Communication Networks," *Proc. Fourth Conf. Data Comm. Systems and Their Performance*, Barcelona, June 1990.

[23] J.W.S. Liu, K.-J. Lin, W.-K. Shih, A.C.-S. Yu, C. Chung, J. Yao, and W. Zhao, "Algorithms for Scheduling Imprecise Computations," *Computer*, vol. 24, no. 5, pp. 58–68, May 1991.

[24] W.-K. Shih, J. Liu, and C. Jen-Yao, "Algorithms for Scheduling Imprecise Computations with Timing Constraints," *SIAM J. Computing*, vol. 20, no. 3, pp. 537–552, June 1991.
[25] W.-K. Shih and J.W.S. Liu, "On-Line Scheduling of Imprecise Computations to Minimize Error," *Proc. IEEE Real-Time Systems Symp.*, Los Alamitos, Calif., Dec. 1992.
[26] B.G. Kim and D. Towsley, "Dynamic Flow Control Protocols for Packet-Switching Multiplexers Serving Real-Time Multipacket Messages," *IEEE Trans. Comm.*, vol. 4, pp. 348–356, Apr. 1986.
[27] E.K.P. Chong and W. Zhao, "Performance Evaluation of Scheduling Algorithms for Imprecise Computer Systems," *J. Systems and Software*, vol. 15, no. 3, pp. 261–277, July 1991.
[28] W. Zhao, S. Vrbsky, and J.W.S. Liu, "Performance of Scheduling Algorithms for Multi-Server Imprecise Systems," *Proc. Fifth Int'l Conf. Parallel and Distributed Computing and Systems*, Oct. 1992.
[29] S.A. Lippman, "Applying a New Device in the Optimization of Exponential Queueing Systems," *Operations Research*, vol. 23, no. 4, pp. 687–710, 1975.
[30] S.B. Gershwin, "Hierarchical Flow Control: A Framework for Scheduling and Planning Discrete Events in Manufacturing Systems," *Proc. IEEE*, vol. 77, no. 1, pp. 195–209, Jan. 1989.
[31] K.M. Mjelde, *Methods of the Allocation of Limited Resources*. New York: John Wiley & Sons, 1983.
[32] D.G. Luenberger, *Linear and Nonlinear Programming*. Reading, Mass.: Addison-Wesley, 1984.
[33] J.K. Dey, J.F. Kurose, D. Towsley, C.M. Krishna, and M. Girkar, "Efficient On-Line Processor Scheduling for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks," *Proc. ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems*, pp. 217–228, Santa Clara, Calif., May 1993.
[34] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[35] S. Ross, *Stochastic Processes*. New York: Wiley, 1983.



Jayanta K. Dey received a BTech degree in computer science and engineering from the Indian Institute of Technology at Madras, India, in 1990 and an MS in computer science from the University of Massachusetts at Amherst in 1993. He is currently enrolled at the University of Massachusetts as a PhD student in computer science. His research interests are in operating systems and networking support for multimedia, video servers, multimedia databases, computer networks, and real-time systems.

James Kurose received his BA degree in physics from Wesleyan University in 1978 and his MS and PhD degrees in computer science from Columbia University in 1980 and 1984, respectively. In 1984, he joined the computer science faculty at the University of Massachusetts, where he is currently an associate professor. He was a visiting scientist in the Communications Department at IBM Research during the 1990/91 academic year.

Dr. Kurose's current research interests include real-time and multimedia communication, network protocols for parallel machines, operating systems, and modeling and performance evaluation. Dr. Kurose is the past editor-in-chief of *IEEE Transactions on Communications* and of *IEEE/ACM Transactions on Networking*. He has been active on the program committees for both the IEEE Infocom conference and the ACM SIGCOMM conference for a number of years.

He is the five-time recipient of the Outstanding Teacher Award from the National Technological University (NTU) and the recipient of the Outstanding Teacher Award from the College of Science and Natural Mathematics at the University of Massachusetts. He is a member of the IEEE, ACM, Phi Beta Kappa, Eta Kappa Nu, and Sigma Xi.

Don Towsley received the BA degree in physics and the PhD degree in computer science from University of Texas in 1971 and 1975, respectively. From 1976 to 1985, he was a member of the faculty of the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst. He is currently a professor of computer science at the University of Massachusetts. During 1982-1983, he was a visiting scientist at the IBM T.J. Watson Research Center, Yorktown Heights, New York, and, during the year 1989-1990, he was a visiting professor at the Laboratoire MASI, Paris, France. His research interests include high speed networks and multimedia systems.

He is currently an editor of the *IEEE/ACM Transactions on Networking* and on the editorial boards of *Networks* and *Performance Evaluation*. He was a program cochair of the joint ACM SIGMETRICS and PERFORMANCE '92 conference. He was elected a fellow of the IEEE for contributions in the modeling and analysis of computer networks. He is also a member of the ACM, ORSA, and the IFIP Working Groups 6.3 and 7.3.