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

J. K. Dey, James Kurose, Don Towsley

CMPSCI Technical Report 93-09
January 27, 1993

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

Jayanta K. Dey James Kurose Don Towsley
Computer Networks and Performance Evaluation Laboratory
Department of Computer Science
University of Massachusetts
Amherst, MA 01003

January 27, 1993

Abstract

In this work, we introduce a real-time task model, where tasks receive a "reward" that depends on the amount of service received. In this model, tasks have associated deadlines at which they must depart from the system. The task computations are such that the longer they are able to execute before their deadline, the greater the value of their computations, i.e., the tasks have the property that they receive *increasing reward with increasing service (IRIS)*. We focus on the problem of scheduling IRIS tasks in a system in which tasks arrive randomly over time, with the goal of maximizing the average reward accrued per task and per unit time. We describe and evaluate a two-level policy for this system. A top-level algorithm executes each time a task arrives and determines the amount of service to allocate to each task in the absence of future arrivals. A lower-level algorithm, an earliest deadline first (EDF) policy in our case, is responsible for the actual selection of tasks to execute. This two-level policy is evaluated through a combination of analysis and simulation. We observe that it provides nearly optimal performance when the variance in the interarrival times and/or laxities is low and that the performance is more sensitive to changes in the arrival process than the deadline distribution.

Keywords: real-time, on-line, scheduling, reward functions.

*The work reported in this paper was supported by ONR grant N00014-87-K-0796 and by NSF grant CCR-9119922. The authors can be contacted at [dey, kurose, towsley]@cs.umass.edu.

1 Introduction

Classical *hard real-time* systems view a real-time task as a computation which must meet its timing constraint, specified as its deadline. Failure of the system to complete a task and provide its results by its deadline leads to a timing-fault and the entire computation is discarded. The underlying assumption of the hard-real time model is that the entire computation is wasted unless completed by its deadline. Of course, scheduling in a hard-real time task so that every task meets its deadline is difficult, given that processing times of tasks may vary as well as the task computations may dynamically vary depending upon their inputs.

The *IRIS* (Increasing-Reward-with-Increasing-Service) model [5] takes another approach to real-time computations. The assumption in this model is that there is a large class of computations which compute results whose *quality* increases as the computations get more CPU time. When these computations are performed by real-time tasks, approximate answers can be returned by the tasks at its deadline, and, in doing so, the *wastage* resulting from discarded computations in the hard real-time model can be minimized. In real-time image processing and tracking, for example, tasks can return fuzzy images and rough location estimates at its deadline. Other examples of tasks exhibiting IRIS behavior have been referred to in the literature as "anytime algorithms" [1, 3], "real-time approximate processing algorithms" [4], or "segmented algorithms" [12]. Examples include tasks that receive, enhance and transmit audio data or video-image frames, tasks for tracking and control, e.g., autonomous vehicle navigation planning [6], tasks for heuristic search [9], tasks for database query processing [7, 17], and tasks computing traditional iterative-refinement numerical algorithms.

Rather than classify real-time tasks using a binary successful/failed criterion, IRIS real-time tasks are characterized by the fact that the longer they are able to compute (before their deadline), the higher is the quality of their computation. Thus the IRIS real-time system is modeled as a single-server system to which tasks arrive and remain until their deadlines, accruing whatever service they can. Given this new model of real-time computation, a new metric is required to evaluate the quality of computation for a task. This metric is in the form of a *reward function* $f(\cdot)$ associated with every task. This function specifies the *value* of the computation as a function of the amount of time it has been able to execute before its deadline. The reward functions help the system to distinguish between the "more important" tasks and the "less important" tasks. Thus tasks in the IRIS model should be assigned reward functions so as to reflect their relative importance.

Note that in this model there is no explicit notion of "processing time" for a task's computation. The processing time of a task can be easily "encoded" into its reward function by making the reward function flat or marginally-increasing after the "duration" of the task, indicating that the resulting quality of the task does not improve any further. Also it is very difficult to explicitly characterize the length of a computation *a priori*, since it depends on various parameters and is often a function of its inputs. But this model does not preclude designing the reward function of a task so that it depends on its inputs, thus implicitly making the amount of time a task needs to be executed before it produces a certain reward rate depend upon on its inputs.

As in [1, 3], we assume that the reward function of a task is an *arbitrary nondecreasing concave function* of its execution time, reflecting the decreasing *marginal* reward as a task receives more service. That is, while every task has increasing reward with increasing service *IRIS* characteristics, the incremental reward for an incremental amount of computation tends to decrease as a task's accrued service time increases. For example, in many iteratively-refined numerical algorithms, e.g., in the Newton-Raphson's method for finding solutions to equations, the approximate results exhibit a geometric convergence to the solution. Concave reward functions for IRIS real-time tasks thus model geometric convergence of Newton-Raphson's method and other fixed point solution methods.

A significant amount of work has been done using the *imprecise computation technique*, a very similar model [13, 16, 15]. 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 unfinished optional subtask. This model thus corresponds to a special case of the IRIS model in which the tasks have identical and linear reward functions. The scheduling mechanism discussed in this paper can easily be adapted to handle mandatory subtasks, as shown in Section 3.2.6. The IRIS task model is a continuous-time model, whereas the imprecise computation technique model is partly discrete-time (in dividing the task into two subtasks) and partly continuous-time (in allowing any amount of optional subtask to be executed). As pointed out before, the IRIS model does not require explicit processing time of a task, whereas in the imprecise computation technique, tasks have known processing times.

For time-dependent path planning [3], Dean and Boddy consider a similar model where the processing times of tasks are multiples of a fixed time slice. This is a discrete-time version of the IRIS task model. 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.

In a different model of imprecise systems, tasks do not have deadlines, but each provide several versions, a *primary version*, which produces a precise result but requires maximum service time of a task, and several *alternate versions*, which provide poorer quality results within a short time. 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 [8] study this model in the context of real-time message transmissions, while [2] and [18] analyze two-version scheduling disciplines for this model in uniprocessor

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

and multiprocessor systems respectively.

The primary contribution of this paper is the development and evaluation of a two-level scheduling policy for tasks exhibiting IRIS characteristics, in a system with *random* task arrivals. At any point in time the scheduler does not know the future arrival times or deadlines of future tasks. At the top level, the scheduler solves a static optimization problem each time that a task arrives in order to determine how much service to provide each task in the system. The lower-level algorithm is concerned with the actual task scheduling, given the service-time allocations of the top-level policy. Here we use the *earliest deadline first* (EDF) policy.

The appropriate performance metric of any scheduling mechanism in the IRIS framework is the average reward per unit time that it can accrue by executing tasks according to its scheduling policy. This is analogous to the performance metric of minimizing error in the imprecise computation model [13, 16, 15].

The evaluation of our proposed scheduling policy is performed through simulation. In addition, analytic upper bounds on the performance of *any* scheduling algorithm in an IRIS framework are presented. In the course of the evaluation, we study the sensitivity of system performance to changes in the arrival process and laxity distribution. We observe that performance is more sensitive to changes in the arrival process than to changes in the deadline distribution. Finally, the static optimization problem considered at the top-level and its associated solutions may be of independent interest as it generalizes earlier work in this area [13, 16, 15] by allowing nonlinear reward functions.

The remainder of this paper is organized as follows: Section 2 describes the task model, and presents upper bounds on performance. Section 3 describes the proposed scheduling algorithm in detail including a description of a static version of the problem and optimal algorithms for maximizing total rewards of task sets in this static version. Section 4 describes some of the numerical results obtained through simulation. Section 5 concludes the paper with a discussion of possible future research.

2 System Model and Upper Bounds

We consider a single server system. Tasks arrive to the system at arbitrary times $a_1 < a_2 < \ldots$ according to a renewal process with rate λ. Here a_i is the arrival time of the i^{th} task. Associated with task i is a laxity τ_i, $i = 1, 2, \ldots$. Task i is required to leave after its laxity expires, at time $a_i + \tau_i$, which we refer to as the task's deadline. We assume that $\{\tau_i\}_{i=1}^{\infty}$ is a sequence of independent and identically distributed (i.i.d.) random variables (r.v.'s) with mean $E[\tau]$. Also associated with each task is a reward function $f_i : I\!R \rightarrow I\!R$ which is assumed to be nondecreasing and concave. Here $f_i(x)$ is the reward accrued by task i when it receives a total of x units of service. While task i resides in the system, it may receive service according to some policy π. The service time accrued by task i under π is denoted as A_i^π and the associated reward is $R_i^\pi = f_i(A_i^\pi)$ with means $E[A^\pi]$ and $E[R^\pi]$ respectively. The only assumptions made regarding π is that it is *non-anticipative*, i.e., it does not know future arrival times or the deadlines associated with future tasks. Otherwise, it is permitted to take any action including preempting tasks that are in service.

We only consider policies such that $R^\pi = \lim_{i\to\infty} R_i^\pi$ exists. We are interested in the average reward per task, $E[R^\pi]$, and the average accrued reward per unit time, $\gamma^\pi = \lambda E[R^\pi]$. We now derive two upper bounds on the reward rate for any such policy. The first bound requires only that the reward function be independent of the task, i.e., that all tasks be homogeneous: $f_i(x) = f(x)$, whereas the second bound requires that task arrivals be described by a Poisson process. We have,

$$\begin{aligned} E[R^\pi] &= E[f(A^\pi)], \\ &\le f(E[A^\pi]). \end{aligned} \tag{1}$$

where we use Jensen's inequality[14]. In general,

$$E[A^\pi] \le min(1/\lambda, E[\tau]).$$

Since f is nondecreasing, this yields the following bound

$$\gamma^\pi \le \begin{cases} \lambda f(E[\tau]), & E[\tau] < 1/\lambda, \\ \lambda f(1/\lambda), & \text{otherwise.} \end{cases} \tag{2}$$

If the arrival process is Poisson, we can derive an exact expression for $E[A^\pi]$. First, observe that the system can be modeled as an $M/GI/\infty$ queue because tasks remain in the system until their deadlines, irrespective of the number of other tasks in the system. Thus 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 $\rho = \lambda E[\tau]$. Let B denote the length of an $M/G/\infty$ busy period and let N_b denote the number of tasks served in a busy period. We can express $E[A] = E[B]/E[N_b]$. Now $E[B]$ is related to $\Pr[N = 0]$ by the following expression,

$$\Pr[N = 0] = \frac{1/\lambda}{E[B] + 1/\lambda},$$

or

$$E[B] = (e^\rho - 1)/\lambda.$$

The mean number of tasks served in a busy period is given by

$$\begin{aligned} E[N_b] &= \lambda(E[B] + 1/\lambda), \\ &= e^\rho. \end{aligned}$$

Hence we obtain $E[A^\pi] = (1 - e^{-\rho})/\lambda$ which yields the bound

$$\gamma^\pi \le \lambda f((1 - e^{-\rho})/\lambda). \tag{3}$$

Observe that $E[A^\pi]$ does not depend on π and, consequently, cannot distinguish between the performances of two policies.

3 Proposed Scheduling Policy

The primary design problem is to identify policies that provide high reward rates. As the problem of identifying the optimal policy appears to be extremely difficult, we propose a heuristic policy which provides demonstrably good performance. Specifically, we propose a two-level scheduling policy with which to schedule IRIS tasks. The top-level algorithm is executed at the time of a task arrival and is used to determine the amount of service to allocate to tasks until the next task arrival. The lower-level algorithm is concerned with the actual selection of tasks to execute.

In our particular design, the service time allocations are chosen at each arrival time 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 our implementation, the algorithm that solves this problem executes assuming that there will be no future arrivals. As previously discussed, the service time allocations output by the algorithm are used by the lower-level scheduler to determine when to switch from one task to another.

The lower-level policy actually schedules the tasks into service and attempts to provide them with the allocations obtained from the top-level scheduler. Note that the target service allocation may change at each arrival. We have chosen the EDF algorithm as the lower-level policy in our implementation. Numerous alternatives were considered in a previous study[10, 11]. However, the reward rate was demonstrated to be relatively insensitive to the actual choice of the lower-level policy.

3.1 Formulation of the Static Version of the Problem

In this subsection, we develop a static version of the problem where all tasks arrive at the same time, and derive the properties to be satisfied by any algorithm which computes optimal service time allocations for a task set. While such a simultaneous-arrival assumption is restrictive in itself, it can be used as the top-level computation of the two-level scheduling approach described above.

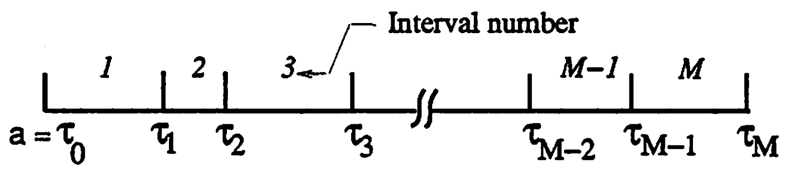

Figure 1: A time interval showing task deadlines and interval numbers

Consider a finite set of M tasks with arrival times $a_1 = a_2 = \ldots = a_M$ and deadlines $\tau_1 \leq \cdots \leq \tau_M$. For notational convenience, define $\tau_0 = a = a_1 = \ldots = a_M$. 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 Figure 1, *interval number j* refers to the interval $(\tau_{j-1}, \tau_j]$. When the i^{th} task departs, if it has received x units of service, it receives a *reward* of $f_i(x)$, with its derivative defined to be $g_i(x) \equiv df_i(x)/dx$. We will find it convenient to define $g_i^{-1}(x)$ to be the inverse function of $g_i(x)$, $i = 1, \ldots, M$. The reward functions are assumed to be continuous and g^{-1} is assumed to be a function in the rest of this paper.

A *schedule* determines the amount of service to be given to each task during each interval. We assume that the CPU is *work-conserving*. 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 a subinterval j is the set of all tasks $\{j, j+1, \ldots, M\}$ which have j as a schedulable interval.

Let $x_{i,j}$ denote the amount of service given to task i ($i = 1, \ldots, M$) during interval $j = 1, \ldots, i$. Observe that $x_{i,j} = 0$ whenever $j > i$. The total amount of service received by task i is defined by $x_i \equiv \sum_{j=1}^{i} x_{i,j}$.

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, P, is:

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

subject to

$$\sum_{i=j}^{M} x_{i,j} \quad = \quad \tau_j - \tau_{j-1}, \quad j = 1, \ldots, M, \tag{4}$$

$$x_{i,j} \geq 0, \quad 1 \leq j \leq i \leq M. \tag{5}$$

The first constraint equates the length of the j^{th} interval to the amount of service given to all of the schedulable tasks in that interval, and reflects the non-idling policy of the processor. The second constraint simply states that a task receives a non-negative amount of service in each of its schedulable intervals.

Thus the problem is a nonlinear optimization problem of a concave objective function with M equality constraints and $M(M-1)/2$ inequality constraints.

Before describing the algorithm to produce an optimal solution to problem P above, we introduce an *auxiliary* optimization problem $P_k, \quad 1 \leq k \leq M$. P_k is a subproblem of P with tasks $k, k+1, \ldots, M$ all having the same arrival time of τ_{k-1}. Problem P_1 is equivalent to problem P. The solution algorithm for P sequentially solves problems $P_i, \quad i = M, \ldots, 1$.

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

subject to

$$\sum_{i=j}^{M} x_{i,j} = \tau_j - \tau_{j-1}, \quad j = k, \ldots, M, \tag{6}$$

$$x_{i,j} \geq 0, \quad k \leq j \leq i \leq M. \tag{7}$$

Appendix A derives several properties satisfied by any optimal solution to P. The following theorem describes the property satisfied by any optimal solution to problem P_i. The top-level algorithms described in the next subsection, utilizes this property to guarantee an optimal solution to problem P.

Define $\bar{x}_j^{(i)}$ as the service time for task j in the optimal solution for problem P_i. Similarly, $\mu_j^{(i)}$ denotes the Langrange multiplier for task j in P_i.

Theorem 1 *Let the optimal solution to problem P_{i+1} allocate service time $\bar{x}_j^{(i+1)}$ to task j. Let K be the set of tasks which receive non-zero allocation in the interval $(\tau_{i-1}, \tau_i]$ in the optimal solution to P_i, i.e., $K = \{k | \bar{x}_k^{(i+1)} < \bar{x}_k^{(i)}\}$. Then the optimal solution to P_i satisfies:*

$$\sum_{k \in K} g_k^{-1}(\mu^{(i)}) = \sum_{k \in K} \bar{x}_k^{(i+1)} + \tau_i - \tau_{i-1}$$

where $\mu^{(i)} = \mu_k^{(i)}$ = the value of the Langrange multiplier for all tasks $k \in K$ in P_i.

PROOF: Since the scheduling policy does not allow idling of the CPU while jobs await service,

$$\sum_{k\in K}\left(x_k^{*(i)} - x_k^{*(i+1)}\right) = \tau_i - \tau_{i-1}$$

or

$$\sum_{k\in K} x_k^{*(i)} = \sum_{k\in K} x_k^{*(i+1)} + \tau_i - \tau_{i-1} \tag{8}$$

From Lemma 1.i and 1.ii in Appendix A we know that all tasks $k \in K$ have the same value of g and $\mu^{(i)}$, and that the rest of the tasks in P_i may have lower g and $\mu^{(i)}$ values. Therefore, equation (8) is rewritten as:

$$\sum_{k\in K} g_k^{-1}(\mu^{(i)}) = \sum_{k\in K} x_k^{*(i+1)} + \tau_i - \tau_{i-1} \tag{9}$$

∎

3.2 Top-Level Scheduling Algorithms

This subsection develops algorithms to compute optimal service time allocations for task sets having different kinds of reward functions. The first two algorithms are for tasks having arbitrary reward functions. The next two algorithms are more efficient versions for tasks having exponential reward functions and tasks having identical reward functions respectively. The final algorithm is for scheduling tasks with piecewise-linear reward functions. Piecewise-linear reward functions are important since arbitrary non-linear functions can be approximated by piecewise-linear functions. Also, a task may have its reward function described by multiple pieces to indicate different regions of operation: a critical portion described by a piece having a high derivative, a concave portion to indicate a task region returning results of non-decreasing quality, and a flat third piece indicating lack of quality improvement of the task result. Note that it is very simple to modify the above algorithms to generate hybrid algorithms for tasks whose reward functions are a medley of the ones described above. Finally, handling of tasks which have a constraint on the minimum amount of service it must receive, is described.

3.2.1 Algorithm for general reward functions, Version 1

The first algorithm, described in Figure 2, uses the results of Theorem 1 to determine the values of x_i that solve problem P. The algorithm solves P_j in the j^{th} iteration in step 2. In this j^{th} 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

```
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 $g_l(x_l)$.
        Using a binary search, find $l$ s.t.
           $\sum_{i=1}^{l+1}[g_{\pi(i)}^{-1}(g_{l+1}(x_{l+1})) - x_{\pi(i)}] > \tau_m - \tau_{m-1} \geq \sum_{i=1}^{l}[g_{\pi(i)}^{-1}(g_l(x_l)) - x_{\pi(i)}]$
        Solve for $\mu$ in
           $\sum_{i=1}^{l} g_{\pi(i)}^{-1}(\mu) = \sum_{i=1}^{l} x_{\pi(i)} + \tau_m - \tau_{m-1}$
        $x_{\pi(i)} = g_{\pi(i)}^{-1}(\mu), \quad i = 1, \ldots, l.$
```

Figure 2: **Algorithm for general reward functions, Version 1.**

these tasks can actually be scheduled using the condition outlined in the previous theorem, i.e., we find a number l, such that l tasks can be allocated in $(\tau_{j-1}, \tau_j]$ but $l+1$ tasks cannot be. The value of μ is then computed using equation (9) from which the values of the $x_i^{(j)}$s are obtained. Note that this algorithm is for the case of arbitrary non-decreasing concave continuous reward functions.

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 $O(M^2 \log M)$.

3.2.2 Algorithm for general reward functions, Version 2

We can improve Algorithm 1 by using sets of tasks instead of individual tasks and maintaining precomputed values of the sums used to find the value of l in the binary search of version 1. The key idea behind this improved version is the observation that equation (16) not only tells us that if two tasks receive service allocations in interval j they have the same derivative in the optimal solution for P, but also that if two tasks have equal derivatives of their reward functions in the optimal solution for problem P_j, they continue to have equal derivatives in solutions for problem $P_{j-1}, P_{j-2}, \ldots, P_1$. Hence their derivatives change simultaneously, if at all, in problems $P_{j-1}, \ldots, P_1$, thereby allowing us to group them into a single set. Formally, during step m, $\mathcal{L} = \{S_1, \ldots, S_r\}$, where $\cup_{i=1}^{r} S_i = \{1, \ldots, m\}$, $S_i \cap S_j = \emptyset, i \neq j$ and the tasks within $S \in \mathcal{L}$ have x_i's such that $g_i(x_i) \equiv g_S, \forall i \in S$. Also, $\forall S_i, S_j \in \mathcal{L}, i > j$ iff $g_{S_i} < g_{S_j}$, i.e., the sets are ordered in $\mathcal{L}$ according to decreasing values of g_S. Define $A_{S_i} \equiv \sum_{j=1}^{i} \sum_{k \in S_j} g_k^{-1}(g_{S_i})$ and $B_{S_i} \equiv \sum_{j=1}^{i} \sum_{k \in S_j} x_k$. The version 2 algorithm is described in Figure 3.

```
1. initialize
    $\mathcal{L} = \emptyset$
    $A_{S_i} = B_{S_i} = 0, \forall i$
2. loop
    for $m = M$ downto 1 do
      $S = \{m\}$
      Insert $S$ into $\mathcal{L}$, merging with some other set if necessary. Assume the insertion position is
      $i, i \in \{1, \ldots, |\mathcal{L}|\}$.
      if $S$ is not merged with any set,
        Compute $A_{S_i} = \sum_{j=1}^{i} \sum_{k \in S_j} g_k^{-1}(g_S)$
      for $j = i$ to $|\mathcal{L}|$ do
      {Update $A_S$'s to reflect inclusion of task $m$}
        $A_{S_j} = A_{S_j} + g_{S_i}^{-1}(g_{S_i})$
      Using a binary search, find $l$ s.t.
            $A_{S_{l+1}} - B_{S_{l+1}} > \tau_m - \tau_{m-1} \geq A_{S_l} - B_{S_l}$
      Merge sets. $S_1' = S_1 \cup S_2 \cup \ldots \cup S_l$
      $A_{S_1'} = A_{S_l}, \quad B_{S_1'} = B_{S_l} + \tau_m - \tau_{m-1}$
      for $j = l + 1$ to $|\mathcal{L}|$ do
        $B_{S_j} = B_{S_j} + \tau_m - \tau_{m+1}$
3. compute $x_i$'s
    for all $S \in \mathcal{L}$ do
      Solve for $\mu$ in $\sum_{i \in S} g_i^{-1}(\mu) = B_S$
      for all $i \in S$ do
        $x_i = g_i^{-1}(\mu)$
```

Figure 3: Algorithm for general reward functions, Version 2.

Since there are at most M sets, the number of set unions possible is at most M. The binary search in each iteration takes $O(\log M)$ time, and the inner **for** loops are $O(M)$. Hence the version 2 algorithm takes time $O(M^2)$, which is an improvement over version 1.

3.2.3 Algorithm for exponential reward functions

If the reward function is $f_i(x) = 1 - e^{-\delta_i(x+a_i)}$, then this procedure can be simplified because

$$g_i(x) = \delta_i e^{-\delta_i(x+a_i)},$$

or

$$g_i^{-1}(\alpha) = (\ln \delta_i - \ln \alpha)/\delta_i - a_i.$$

```
1. initialize
  $\mathcal{L} = \emptyset$
2. loop
  for $m = M$ downto 1 do
    $S = \{m\}$; $A_S = (\ln\delta_m)/\delta_m - a_m$; $B_S = 1/\delta_m$; $X_S = 0$; $g_S = g_m(0)$
    Insert $S$ into $\mathcal{L}$ such that the elements $S_1, \ldots S_r$ are in decreasing order of $g_{S_l}$.
    Update $A_*$ and $B_*$ of the set, if any, where $S$ was inserted
    $j = 0$; $S' = \emptyset$; $A_{S'} = 0$; $B_{S'} = 0$; $x_{S'} = 0$
    do $j = j + 1$; $S' = S' \cup S_j$; $A_{S'} = A_{S'} + A_{S_j}$;
        $B_{S'} = B_{S'} + B_{S_j}$; $x_{S'} = x_{S'} + x_{S_j}$
      until $j = r$
      or
      $A_{S'} + A_{S_{j+1}} - (B_{S'} + B_{S_{j+1}})\ln g_{S_{j+1}} - x_{S'} - x_{S_{j+1}} > \tau_m - \tau_{m-1} \geq A_{S'} - B_{S'}\ln g_{S_j} - x_{S'}$
      $x_{S'} = x_{S'} + \tau_m - \tau_{m-1}$
      $g_{S'} = \exp([A_{S'} - x_{S'}]/B_{S'})$;
3. calculate $x_i$'s
    for $l = 1$ to $r$ do
      for $i \in S_l$ do
        $x_i = (\ln\delta_i - \ln g_{S_l})/\delta_i - a_i$
```

Figure 4: Algorithm for exponential reward functions.

Hence the partial sum $\sum_{i \in S} g^{-1}(\alpha)$ can be expressed as

$$\sum_{i \in S} g_i^{-1}(\alpha) = A_S - B_S \ln\alpha,$$

where $S \subset \{1, \ldots, M\}$ and $A_S = \sum_{i \in S}(\ln\delta_i/\delta_i - a_i)$, $B_S = \sum_{i \in S}\delta_i^{-1}$.

The algorithm for this case is shown in Figure 4. Here during step m, $\mathcal{L} = \{S_1, \ldots, S_r\}$ where $\cup_{i=1}^{r} S_i = \{1, \ldots, m\}$, $S_i \cap S_j = \emptyset$, $i \neq j$ and the tasks within $S \in \mathcal{L}$ have x_i's such that $g_i(x_i) \equiv g_S$, $\forall i \in S$. Last, $X_S \equiv \sum_{i \in S} x_i$. We recompute A_S and B_S on the fly whenever we update (add or remove elements) from set S.

It is clear from this algorithm, that the complexity is now $O(M \log M)$ due to the fact that there are M steps in the outer **for** loop and that the insertion of $\{m\}$ into $\mathcal{L}$ is $O(\log M)$.

3.2.4 Algorithm for identical reward functions

When all the tasks have identical reward functions, a linear-time service-time balancing algorithm shown in Figure 5 suffices. The algorithm maintains sets of tasks such that elements of each set have identical service time allocations, and no two sets have tasks with the same allocation. During step m of the

```
1. initialize
   Initialize $\mathcal{L}$ with a dummy set $S$ having a single element 0 with $x_0 = \infty$, $A_S = \infty$
2. loop
   for $m = M$ downto 1 do
      $remainder = \tau_m - \tau_{m-1}$
      $S' = \{m\}, A_{S'} = 0$
      Attach $S'$ at front of $\mathcal{L}$
      while ($remainder > 0$) do
         Take next set $S_\alpha \in \mathcal{L}$; Let $S_\beta$ be the set after $S_\alpha \in \mathcal{L}$.
         $allocreqd = A_{S_\beta} - A_{S_\alpha}$
         if ($remainder \geq allocreqd$)
         {give elements of $S_\alpha$ equal service as elements of $S_\beta$}
            $remainder = remainder - allocreqd$
            Unify sets $S_\alpha$ and $S_\beta$ to $S^*$. Compute $A_{S^*} = \frac{|S_\alpha|+|S_\beta|}{|S_\beta|} \cdot A_{S_\beta}$
         else
         {elements in $S_\alpha$ get all of remainder}
            $A_{S_\alpha} = A_{S_\alpha} + remainder$
               $remainder = 0$
3. calculate $x_i$'s
      for each $S \in \mathcal{L}$ do
         for each $i \in S$ do
            $x_i = A_S/|S|$
```

Figure 5: **Algorithm for identical reward functions.**

algorithm, $\mathcal{L} = \{S_1, \ldots, S_r\}$ where $\cup_{i=1}^{r} S_i = \{1, \ldots, m\}$, $S_i \cap S_j = \emptyset$, $i \neq j$ and the tasks within $S \in \mathcal{L}$ have x_i's such that $x_i \equiv x_S$, where x_S is the service time allocated to all tasks in set S. The list $\mathcal{L}$ is ordered as follows: if S_α precedes S_β in $\mathcal{L}$, then $x_{S_\alpha} < x_{S_\beta}$. We define $A_S = \sum_{i \in S} x_i = |S| \cdot |x_S|$.

The maximum possible number of sets in $\mathcal{L}$ is M. Thus, at most M set unifications are possible in one execution of the algorithm, which corresponds to the **if** part inside the **while** loop. The **else** part within the **while** loop can therefore be executed at most M times in the entire execution of the algorithm. Thus this algorithm's running time is linear.

However, when we use this algorithm as the top-level algorithm in the two-level scheduling approach, its running time is $O(M \log M)$. This results from the fact that when the algorithm is executed at an arrival instance, there can be other tasks in the system which have earlier arrival times and may have received some service prior to the execution of this algorithm. Hence we cannot always attach a task to the front of list $\mathcal{L}$ as is done in the static version. A binary search has to be performed on $\mathcal{L}$ to find the appropriate insertion place; this raises the complexity to $O(M \log M)$.

We argue that this latter algorithm provides the same solution as that given by the general algorithm

```
1. initialize
   $\mathcal{L} = \emptyset$ 2. loop
   for $m = M$ downto 1 do
      $remainder = \tau_m - \tau_{m-1}$
      $S' = \{m\}$
      Insert $S'$ into $\mathcal{L}$ such that the above described order is maintained
      while ($remainder > 0$) do
         Take next set $S \in \mathcal{L}$;   $A_S = \sum_{k \in S}(right_k - x_k)$
         if ($remainder \geq A_S$)
            $remainder = remainder - A_S$
            for each job $k \in S$ do
               $x_k = right_k$
               Using $g_k$ of next tuple, insert $k$ into $\mathcal{L}$ at appropriate position
         else
            $remainder = 0$
            for each job $k \in S$ do
               $x_k = x_k + remainder/|S|$
               if ($x_k == right_k$), using $g_k$ of next tuple, insert $k$ into $\mathcal{L}$ at
                  appropriate position
```

Figure 6: **Algorithm for piecewise-linear reward functions**

when the reward functions are identical. The sets are ordered in increasing values of x, which is the same as ordering them in decreasing order of reward function derivatives g, as in the general algorithm. In the general algorithm, a binary search is used to find the sets which need to be merged to indicate that they have the same value of g after merging. In this latter algorithm, sets are merged to indicate that the tasks in the merged sets have the same value of x after merging. Hence the algorithm gives the optimal solution.

3.2.5 Algorithm for piecewise-linear reward functions

For tasks with piecewise-linear reward functions, we assume that each piece-wise linear function is described as a sequence of two tuples, each tuple corresponding to a linear-segment. The tuple elements for a function are the slope of a constituent segment and the right end point of the segment on the x-axis. Thus each two tuple for job i is of the form $(g_i, right_i)$, and each function is described as a sequence of these tuples, ordered by increasing $right_i$ values. As in the previous algorithms, during step m, $\mathcal{L} = \{S_1, \ldots, S_r\}$ where $\cup_{i=1}^{r} S_i = \{1, \ldots, m\}$, $S_i \cap S_j = \emptyset, i \neq j$ and the jobs within $S \in \mathcal{L}$ have x_i's such that $g_i(x_i) \equiv g_S$. Define $A_S = \sum_{i \in S}(right_i - x_i)$. Figure 6 describes the algorithm.

Let p_k be the number of linear pieces for the reward function for job k. An insertion into $\mathcal{L}$ takes time $O(\log M)$, and there are at most $\sum_{k=1}^{M} p_k$ insertions into the list. A linear segment is accessed at

```
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
        $f_i^*(x) = f_i(x + m_i)$
```

Figure 7: **Transforming tasks with mandatory portions into tasks without them**

most M times. This occurs if the right-end of the segment of a reward function is never reached. Thus the complexity of this algorithm is $O(\log M(\sum_{k=1}^{M} p_k + M))$. When there are a bounded number of segments describing a reward function, the algorithm is $O(M \log M)$.

3.2.6 A Note on handling tasks with mandatory subtasks

The above algorithms can easily handle, with a slight modification, tasks having constraints on the minimum amount of service each must receive. The algorithm described in Figure 7 describes the algorithm which transforms the problem into one where the deadlines of the tasks have been reduced in a way to account for the mandatory portions. In the algorithm, m_i denotes the length of the mandatory subtask of task i.

3.3 The Lower-Level Policy

As indicated before, the purpose of the lower-level policy is to actually schedule tasks. We chose EDF as the second level policy because, in addition to the reasons described earlier, in the case when there are no arrivals, or in the case when deadlines of all the tasks in the system are before the time of the next arrival, this policy gives the minimum number of context switches.

Each of the top-level algorithms could be modified to determine which task to schedule in each subinterval. But the resultant schedule would have several more context switches, a significant overhead in real-time systems. Also by using this two-level scheduling mechanism, we separate the issues of how much service to allocate to each task versus which task to schedule at the next scheduling instance. By separating

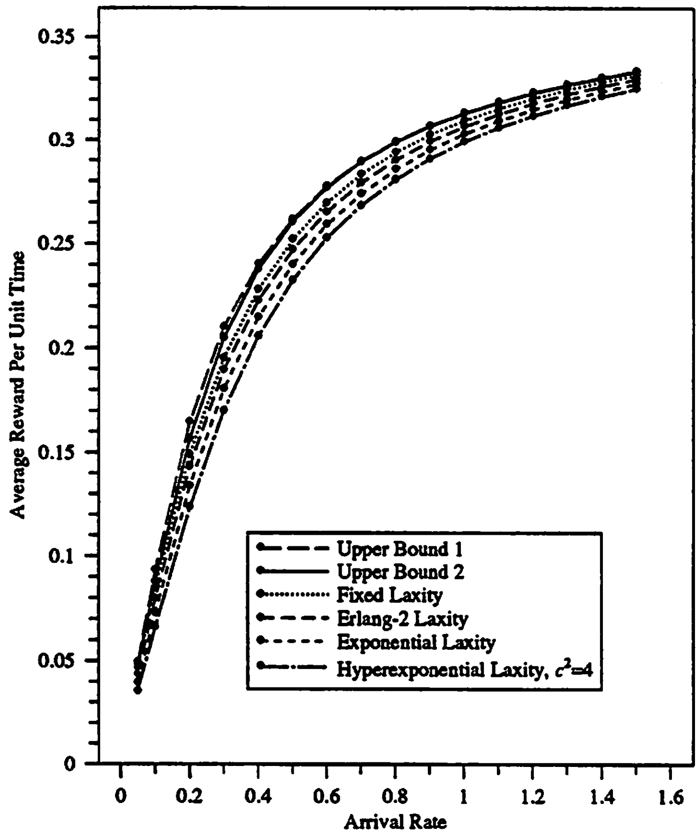

Figure 8: Reward per unit time for Poisson arrivals and different laxity distributions, $\delta = 0.4$

these concerns, algorithms at each level are kept simple, and it allows us to use independent algorithms at either level.

4 Numerical Results

To evaluate the performance of our two-level scheduling approach, we simulated the algorithm for tasks which have identical reward functions. The reward function is $f(x) = 1 - e^{-\delta x}$. We varied three parameters: the interarrival distribution, the laxity distribution, and δ. The simulations have been done for arrival rates $0.05, 0.1, 0.2, \ldots, 1.5$, with tasks having mean laxity 10, i.e. the system is modeled at both high and low rates. We simulated 1 million tasks and the 95% confidence interval widths for all the results were found to be between $4 - 5\%$ of the point value.

Figure 8 shows the performance of the algorithm for $\delta = 0.4$ and a variety of laxity distributions, when the task arrivals are Poisson. Upper Bound 1 in the figure is the looser upper bound derived in equation

(2), and Upper Bound 2 is the tighter upper bound for Poisson task arrivals, as derived in equation (3). Hence equation (2) can be rewritten for the above reward function as $\lambda(1 - e^{(-\delta/\lambda)})$. Similarly, equation (3) for the exponential reward function is $\lambda(1 - e^{\frac{-\delta(1-e^{-\rho})}{\lambda}})$. The two upper bounds asymptotically converge to δ, and in the figure, the upper bound 1 converges with the upper bound 2 as λ increases.

We note that the average reward per unit time comes closest to the upper bounds when the tasks have a fixed laxity and is farthest when the laxity distribution is hyperexponential, i.e. as the variability (coefficient of variation, c.v.) of the laxity increases, the average reward per unit time decreases. In the case of hyperexponential laxities, the tasks with larger laxities get greater service on the average than the tasks with smaller laxities, thus bringing down the reward per unit time, since the reward function is concave. It is also interesting to note that the curves come close to the upper bounds for very low and high values of the arrival rate, but for intermediate values, the difference from the optimum is noticeable. We conjecture that this is explained by the fact that for any value of δ, the reward curve can roughly be divided into three regions. In the first region (the closest to the origin), the reward curve is linear; in the third region, the curve flattens out as it reaches its asymptotic value and thus again is linear. In the middle region, the curve is nonlinear and it is here that the decreasing marginal rewards become more significant. Hence it is in this region that the difference in performance is greatest for different laxity distributions. Note that when the arrival rate is very high, most of the tasks receive very little service, thus falling in the first region where the reward is approximately linear. Hence the marginal reward per task differs little for different laxity distributions. For low arrival rates, almost all tasks receive service equal to their laxity, since almost all busy periods consist of only one task, resulting in the small differences in the curve.

Figure 9 plots the average reward rate for $\delta = 0.8$. We note that the behavior of the reward curve for $\delta = 0.8$ is similar to that for $\delta = 0.4$, except that the elbow, i.e., the middle region, as well as the third region where the reward curve flattens out, occurs at higher values of the arrival rate. Otherwise, the nature of differences for different laxity distributions as well as the difference from the exact upper bound remain the same.

Figures 10 and 11 contain performance results for tasks having identical laxity distributions, fixed and hyperexponential respectively, and different interarrival distributions. From these figures, we note that the average service is more sensitive to the interarrival process, than to the laxity distribution. The differences in average reward per unit time between exponential interarrival times and Erlang-2 interarrival times is small, indicating that for small coefficients of variation, the average rewards per unit time are close to each other, and close to the upper bound. Notice that the average reward per unit time is lowest for hyperexponential interarrival durations. This can be explained as follows. Intuitively, the mean busy period of a $G/G/\infty$ shrinks as the coefficient of variation increases. When the system has hyperexponential

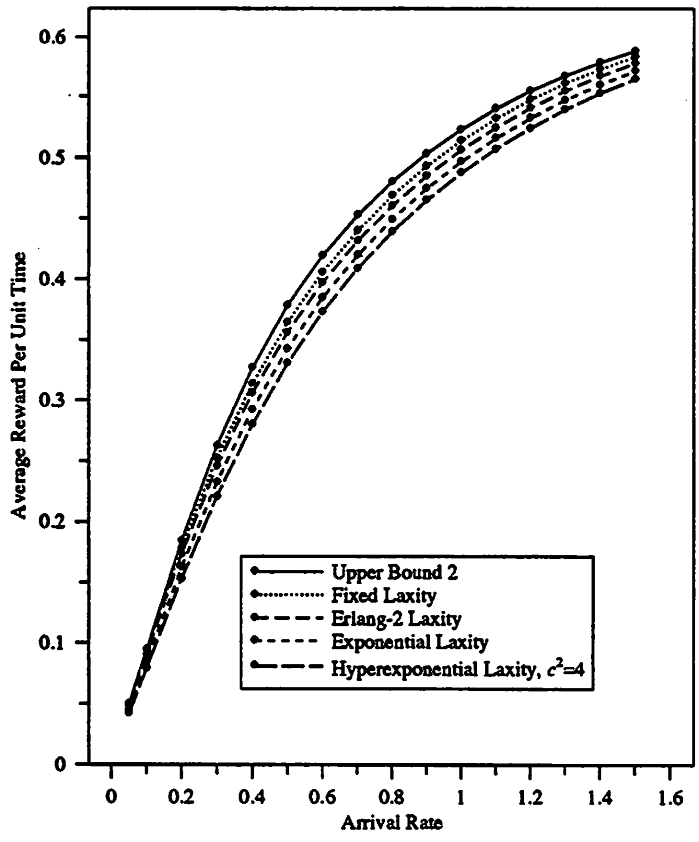

Figure 9: **Reward per unit time for Poisson arrivals and different laxity distributions,** $\delta = 0.8$

arrivals, a burst of arrivals occur within a short interval of time which is followed by a long idle period. Thus the system behaves as though it experiences bulk arrivals followed by a large idle period, reducing the average reward per unit time. A similar argument explains why the average reward per unit time for tasks with hyperexponential laxity in Figure 11 is lower than for tasks with fixed laxity in Figure 10, for the same interarrival distributions.

5 Conclusion

We have developed a two-level scheduling policy for IRIS tasks. We have presented an optimization algorithm for the static version of the problem and then used the algorithm as a heuristic for the general problem. We evaluated our approach by comparing simulation results against the theoretical upper bounds we developed for any scheduling policy of IRIS tasks and observed that our policy provides nearly optimal

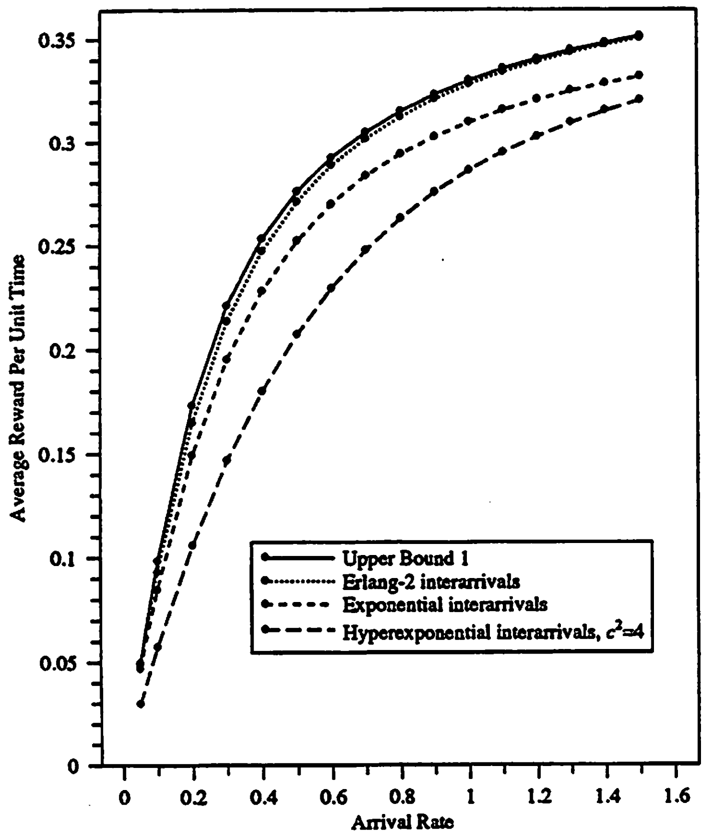

Figure 10: **Reward per unit time for different interarrival distributions and fixed laxity**

performance when the variance in interarrival time and/or laxities is small. The performance is more sensitive to changes in the arrival process than to the deadline process.

Extensions of this work include scheduling IRIS tasks with non-identical reward functions, and investigating other lower-level policies for our two-level scheduling approach.

A Properties of any optimal solution to problem P

The optimal solution to P maximizes the reward rate for the set of tasks. Intuitively, to maximize the reward rate for a set of tasks, in any subinterval we must start with the task with the highest marginal reward, allocate it service till the marginal reward of the task equals that of the task with the second-highest marginal reward, and then allocate both of them service till both their marginal rewards equal that of the task with the third-highest marginal reward, and so on. This process must continue till all the time in the subinterval is

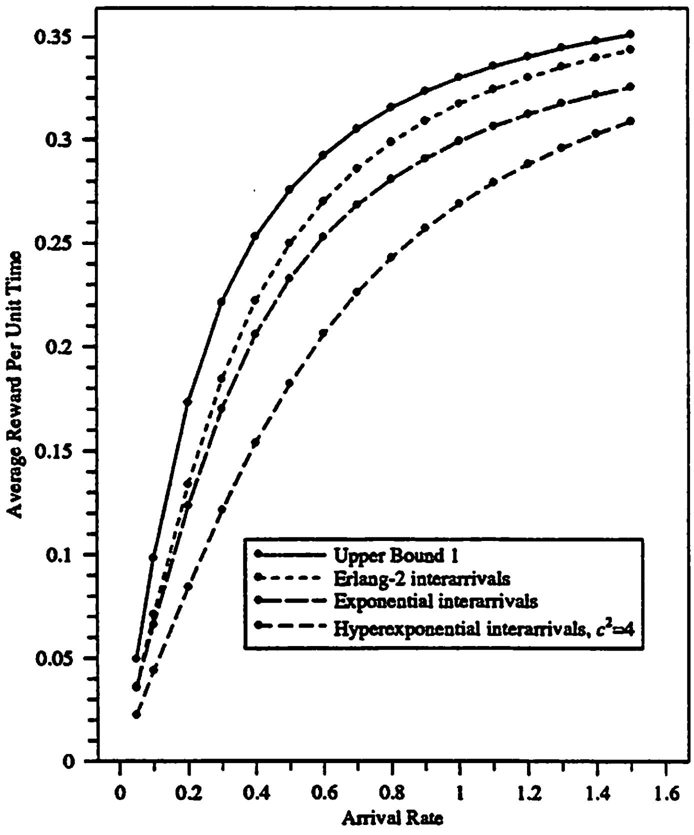

Figure 11: Reward per unit time for different interarrival distributions and H_2 laxity with $c^2 = 4$

scheduled. Thus the tasks which are not scheduled are those with equal or lower marginal rewards. The rest of this section formally proves this intuition by establishing properties of any optimal solution to problem P.

The Langrangian of problem P is given by:

$$L(\mathbf{x}, \mu, \nu) = -\sum_{i=1}^{M} f_i \left(\sum_{k=1}^{i} x_{i,k} \right) + \sum_{k=1}^{M} \mu_k \cdot \left(\sum_{i=k}^{M} x_{i,k} - \tau_k + \tau_{k-1} \right) + \sum_{i=1}^{M} \sum_{k=1}^{i} \nu_{i,k} \cdot x_{i,k},$$

where $\mu_k, \nu_{i,k}$ are the Langrange multipliers.

Any optimal solution to P satisfies the Kuhn-Tucker conditions,

$$\frac{\partial L}{\partial x_{i,j}} = -\frac{\partial f_i \left(\sum_{k=1}^{i} x_{i,k} \right)}{\partial x_{i,j}} + \mu_j + \nu_{i,j} = 0, \quad 1 \le j \le i \le M, \tag{10}$$

$$\frac{\partial L}{\partial \mu_j} = \sum_{i=j}^{M} x_{i,j} - \tau_j + \tau_{j-1} = 0, \quad j = 1, \ldots, M, \tag{11}$$

$$x_{i,j} \geq 0, \qquad \nu_{i,j} \cdot x_{i,j} = 0, \qquad \nu_{i,j} \leq 0, \quad 1 \leq j \leq i \leq M. \tag{12}$$

Equation (10) can be rewritten as

$$-\frac{\partial f_i(x_i)}{\partial x_i} + \mu_j + \nu_{i,j} = 0, \quad 1 \leq j \leq i \leq M \tag{13}$$

or

$$-g_i(x_i) + \mu_j + \nu_{i,j} = 0, \quad 1 \leq j \leq i \leq M \tag{14}$$

From condition (12), we note that $x_{i,j}$ can either be zero or positive. We consider each case separately.

Case 1. $x_{i,j} = 0$. Task i receives no service in subinterval $(\tau_{j-1}, \tau_j]$.

It follows from equation (12) that $\nu_{i,j} \leq 0$. Therefore from equation (14):

$$g_i(x_i) \leq \mu_j, \quad 1 \leq j \leq i \leq M, \quad x_{i,j} = 0 \tag{15}$$

Case 2. $x_{i,j} > 0$. Task i receives service in subinterval $(\tau_{j-1}, \tau_j]$.

In this case, we have $\nu_{i,j} = 0$. Therefore from equation (14):

$$g_i(x_i) = \mu_j, \quad 1 \leq j \leq i \leq M, \quad x_{i,j} > 0 \tag{16}$$

Equation (16) indicates that tasks receiving a non-zero 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 may have a lower value of their reward function derivative as shown in equation (15). Serving these might provide a lower marginal reward.

Let x_i^* be the amount of service given 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 $x^* = [x_{1,1}^*, x_{1,2}^* \ldots, x_{M,M}^*]$ to be an optimal solution to P. Let be the set of all optimal solutions to P.

Lemma 1 *Any $x^* \in$ satisfies the relations,*

i. *if $\exists$ interval k s.t. $x_{i,k}^* > 0$ and $x_{j,k}^* > 0$, then $g_i(x_i^*) = g_j(x_j^*)$, $\quad 1 \leq k \leq i, j \leq M$.*

ii. *if $\exists$ interval k s.t. $x_{i,k}^* > 0$ and $x_{j,k}^* = 0$, then $g_i(x_i^*) \geq g_j(x_j^*)$, $\quad 1 \leq k \leq i, j \leq M$.*

iii. *if $x_j^* = 0$, then $\forall i \in 1, 2, \ldots, (j-1)$ s.t. $x_i^* > 0$, $\quad g_i(x_i^*) \geq g_j(x_j^*)$.*

iv. *if $\exists$ task k s.t. $x_{k,i}^* > 0$ and $x_{k,j}^* > 0$, then $\mu_i = \mu_j$, $\quad 1 \leq i, j \leq k \leq M$.*

PROOF. i. Obtained by applying equation (16) to tasks i and j in interval k.
ii. Obtained by using equation (15) with task j in interval k, and using equation (16) with task i in interval k.
iii. Since $x_i^* > 0$, $\exists$ some interval $k \leq i$, s.t. $x_{i,k}^* > 0$. Since $i < j$, $k < j$. Thus the proof follows from part ii. of this lemma.
iv. This property is obtained by using equation (16) for intervals i and j with task k. ∎

Lemma 2 *For every $x^* \in$:*
i. *The Lagrange multipliers μ_i form a monotonically non-increasing sequence, i.e., $\mu_i \geq \mu_{i+1}$, $1 \leq i < M$,*
ii. *x^* can be constructively transformed into $x'^* \in$, such that for all of the tasks i that receive service in x^*, the corresponding g_i's in x'^* are monotonically non-increasing, i.e., $g_i \geq g_j$, $1 \leq i < j \leq M$, $x_i^*, x_j^* > 0$.*

PROOF. i. Assume the lemma is not true, i.e., $\mu_i < \mu_{i+1}$ for some i. From Lemma 1.iv, 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 equations (15) and (16), we get

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

This implies $\mu_{i+1} \leq \mu_i$, which is a contradiction.
ii. We construct x'^* from x^* by an iterative procedure which operates on every pair of tasks which have received non-zero service in x^*.
Take two tasks i and j, $i < j$, that have received non-zero service in x^*. Let $m = \max\{k | x_{i,k} > 0\}$, and $n = \max\{k | x_{j,k} > 0\}$. The construction is divided into three cases.

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

Case 2. $m < n$. Then we get $g_i(x_i^*) \geq g_j(x_j^*)$ by using equation (16) and Lemma 2.i. 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 m^{th} and n^{th} intervals as follows:

$$x_{i,m}'^* = x_{i,m}^* - min(x_{i,m}^*, x_{j,n}^*), \tag{17}$$

$$x_{j,m}'^* = x_{j,m}^* + min(x_{i,m}^*, x_{j,n}^*), \tag{18}$$

$$x'^*_{i,n} = x^*_{i,n} + min(x^*_{i,m}, x^*_{j,n}), \tag{19}$$

$$x'^*_{j,n} = x^*_{j,n} - min(x^*_{i,m}, x^*_{j,n}). \tag{20}$$

Now, if $x'^*_{i,m} > 0$, $\max\{k|x'^*_{i,k} > 0\} = \max\{k|x'^*_{j,k} > 0\} = m$, and case 1 can now be applied. If $x'^*_{i,m} = 0$, $\max\{k|x'^*_{i,k} > 0\} < \max\{k|x'^*_{j,k} > 0\} = m$, and case 2 can now be applied.

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

Thus Lemma 2 proves the informal intuition discussed at the beginning of this section.

References

[1] BODDY, M., AND DEAN, T. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)* (Detroit, Mi, Aug. 1989), IJCAI-89, pp. 979–984.

[2] CHONG, E. K., AND ZHAO, W. Performance evaluation of scheduling algorithms for imprecise computer systems. *Journal of Systems and Software 15*, 3 (July 1991), 261–277.

[3] DEAN, T., AND BODDY, M. An analysis of time-dependent planning. In *Proceedings of the Seventh National on Artificial Intelligence (AAAI-88)* (Detroit, Mi, Aug. 1988), AAAI-88, pp. 49–54.

[4] DECKER, K., V.R., L., AND R.C., W. Extending a blackboard architecture for approximate processing. *The Journal of Real-Time Systems 2* (1990), 47–79.

[5] DEY, J. K., KUROSE, J. F., TOWSLEY, D., KRISHNA, C. M., AND GIRKAR, M. Efficient on-line processor scheduling for a class of IRIS (Increasing Reward with Increasing Service) real-time tasks. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems* (Santa Clara, California, May 1993), ACM.

[6] HAYES-ROTH, B. Architectural foundations for real-time performance in intelligent agents. *The Journal of Real-Time Systems 2* (1990), 99–125.

[7] HOU, W., OZSOYOGLU, G., AND B.K., T. Processing aggregate relational queries with hard time constraints. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data* (June 1989), ACM.

[8] KIM, B., AND TOWSLEY, D. Dynamic flow control protocols for packet-switching multiplexers serving real-time multipacket messages. *IEEE Transactions on Communications, COM-34 4* (Apr. 1986), 348–356.

[9] KORF, R. E. Depth-limited search for real-time problem solving. *The Journal of Real-Time Systems 2* (1990), 7–24.

[10] KUROSE, J., KRISHNA, C., AND GIRKAR, M. On scheduling real-time tasks with increased-value-with-increased-execution-time characteristics. Unpublished Report, Oct. 1991.

[11] KUROSE, J., TOWSLEY, D., AND KRISHNA, C. Design and analysis of processor scheduling policies for real-time systems. In *Foundations in Real-Time Computing*, A. v. Tilborg, Ed. Kluwer Academic Publishers, 1991, ch. 3, pp. 63–89.

[12] LIU, J. Timing constraints and algorithms. In *Report on the Embedded AI Languages Workshop* (Nov. 1988), University of Michigan, pp. 9–11.

[13] LIU, J. W., LIN, K.-J., SHIH, W.-K., YU, A. C.-S., CHUNG, J.-Y. C., AND ZHAO, W. Algorithms for scheduling imprecise computations. *IEEE Computer 24*, 5 (May 1991), 58–68.

[14] ROSS, S. *Stochastic Processes.* Wiley, New York, 1983.

[15] SHIH, W. K., AND LIU, J. On-line scheduling of imprecise computations to minimize error. In *IEEE Real-Time Systems Symposium* (Los Alamitos, CA, Dec. 1992), IEEE.

[16] SHIH, W. K., LIU, J., AND J.Y., C. Algorithms for scheduling imprecise computations with timing constraints. *SIAM Journal on Computing 20*, 3 (June 1991), 537–552.

[17] SMITH, K., AND LIU, J. Monotonically improving approximate answers to relational algebra queries. In *Proceedings of Compsac* (Sept. 1989).

[18] ZHAO, W., VRBSKY, S., AND LIU, J. Performance of scheduling algorithms for multi-server imprecise systems. In *Proceedings of the 5th International Conference on Parallel and Distributed Computing and Systems* (Oct. 1992).