Optimal Reward-Based Scheduling of Periodic Real-Time Tasks *

Hakan Aydın, Rami Melhem, and Daniel Mossé
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260
{aydin,mosse,melhem}@cs.pitt.edu

Pedro Mejía-Alvarez †
CINVESTAV-IPN. Sección de Computación
Av. I.P.N. 2508, Zacatenco.
México, DF. 07300
pmejia@cs.pitt.edu

Abstract

Reward-based scheduling refers to the problem in which there is a reward associated with the execution of a task. In our framework, each real-time task comprises a mandatory and an optional part, with which a nondecreasing reward function is associated. Imprecise Computation and Increased-Reward-with-Increased-Service models fall within the scope of this framework. In this paper, we address the reward-based scheduling problem for periodic tasks. For linear and concave reward functions we show: (a) the existence of an optimal schedule where the optional service time of a task is constant at every instance and (b) how to efficiently compute this service time. We also prove that RMS-h (RMS with harmonic periods), EDF and LLF policies are optimal when used with the optimal service times we computed, and that the problem becomes NP-Hard, when the reward functions are convex. Further, our solution eliminates run-time overhead, and makes possible the use of existing scheduling disciplines.

1 Introduction

In a real-time system each task must complete and produce correct output by the specified deadline. However, if the system is overloaded it is not possible to meet each deadline. In the past, several techniques have been introduced by the research community regarding the appropriate strategy to use in overloaded systems of periodic real-time tasks.

One class of approaches focuses on providing somewhat less stringent guarantees for temporal constraints. In [11], some instances of a task are allowed to be skipped entirely. The *skip factor* determines how often instances of a given task may be left unexecuted. A best effort strategy is introduced in [8], aiming at meeting k deadlines out of n instances of a given task. This framework is also known as *(n,k)-firm deadlines* scheme. Bernat and Burns present in [2] a hybrid and improved approach to provide hard real-time guarantees to k out of n consecutive instances of a task.

The techniques mentioned above tacitly assume that a task's output is of no value if it is not executed completely. However, in many application areas such as multimedia applications [17], image and speech processing [4, 6, 19], time-dependent planning [3], robot control/navigation systems [21], medical decision making [9], information gathering [7], real-time heuristic search [12] and database query processing [20] a partial or approximate but timely result is usually acceptable.

The *Imprecise Computation* [5, 15] and *IRIS (Increased Reward with Increased Service)* [10, 13] models were proposed to enhance the resource utilization and provide graceful degradation in real-time systems. In these models, every real-time task is composed of a mandatory part and an optional part. The former should be completed by the task's deadline to provide output of minimal quality. The optional part is to be executed after the mandatory part while still before the deadline, if there are enough resources in the system that are not committed to running mandatory parts for any task. The longer the optional part executes, the better the quality of the result (the higher the reward).

The algorithms proposed for imprecise computation applications concentrate on a model that has an upper bound on the execution time that could be assigned to the optional part [5, 15, 18]. The aim is usually to minimize the (weighted) sum of errors. Several efficient algorithms are proposed to solve optimally aperiodic scheduling problem of imprecise computation tasks [15, 18]. A common assumption in these studies is that the quality of the results produced is a *linear* function of the precision error; consequently, the possibility of having more general error functions is usually not addressed.

An alternative model allows tasks to get increasing reward with increasing service (IRIS model) [10, 13] without an upper bound on the execution times of the tasks (though the deadline of the task is an implicit upper bound) and without the separation between mandatory and optional parts [10]. A task executes for as long as the scheduler allows before its deadline. Typically, a nondecreasing *concave* reward function is associated with each task's execution time. In [10] the problem of

* This work has been supported by the Defense Advanced Research Projects Agency through the FORTS project (Contract DABT63-96-C-0044).

† Work done while at the University of Pittsburgh

maximizing the total reward in a system of aperiodic independent tasks is addressed. The optimal solution with static task sets is presented, as well as two extensions that include mandatory parts and policies for dynamic task arrivals.

Note that imprecise computation and IRIS models are closely related, since the performance metrics can be defined as duals (maximizing the total reward is a dual of minimizing the total error). Similarly, a concave reward function corresponds to a convex error function, and vice versa.

We use the term "Reward-based scheduling" to encompass scheduling frameworks such as Imprecise Computation and IRIS models, where each task can be decomposed into mandatory and optional subtasks. A nondecreasing reward function is associated with the execution of each optional part.

An interesting question concerns types of reward functions which represent realistic application areas. A *linear* reward function [15] models the case where the benefit to the overall system increases *uniformly* during the optional execution. Similarly, a *concave* reward function [10, 13] addresses the case where the greatest increase/refinement in the output quality is obtained during the first portions of optional executions. The first derivative of a nondecreasing concave function is nonincreasing. Linear and general concave functions are considered the most realistic and typical in the literature since they adequately capture the behavior of many application areas such as those mentioned above [4, 6, 19, 3, 21, 12, 7, 17, 20]. In this paper, we show that the case of *convex* reward functions is an NP-Hard problem and thus focus on linear and concave reward functions. Reward functions with 0/1 constraints, where no reward is accrued unless the *entire* optional part is executed, or step reward functions have also received some interest in the literature. Unfortunately, this problem has been shown to be NP-Complete in [18].

Periodic reward-based scheduling remains relatively unexplored, since the important work of Chung, Liu and Lin [5]. In that paper, the authors classified the possible application areas as "error non-cumulative" and "error cumulative". In the former, errors (or optional parts left unexecuted) have no effect on the future instances of the same task. Well-known examples of this category are tasks that periodically receive, process and transmit audio, video or compressed images [4, 6, 19] as well as information retrieval tasks [7, 20]. In "error cumulative" applications, such as radar tracking, an optional instance must be executed completely at every (predetermined) k invocations. The authors further proved that the case of error-cumulative jobs is an NP-Complete problem. In this paper, we restrict ourselves to error non-cumulative applications.

Recently, a QoS-based resource allocation model (QRAM) has been proposed for periodic applications [17]. In that study, the problem is to optimally allocate several resources to the various applications such that they simultaneously meet their minimum requirements along multiple QoS dimensions and the total system utility is maximized. In one aspect, this can be viewed as a generalization of optimal CPU allocation problem to multiple resources and quality dimensions. Further, dependent and independent quality dimensions are separately addressed for the first time in this work. However, a fundamental assumption of that model is that the reward functions and resource allocations are in terms of *utilization of resources*. Our work falls rather along the lines of Imprecise Computation model, where the reward accrued has to be computed separately over all task instances and the problem is to find the optimal service times for *each* instance and the optimal schedule with these assignments.

Aspects of Periodic Reward-Based Scheduling Problem

The difficulty of finding an optimal schedule for a periodic reward-based task set has its origin on two objectives that must be simultaneously achieved, namely: (i) Meeting deadlines of mandatory parts at *every* periodic task invocation, and (ii) Scheduling optional parts to maximize the total (or average) reward.

These two objectives are both important, yet often incompatible. In other words, hard deadlines of mandatory parts may require sacrificing optional parts with greatest value to the system. The analytical treatment of the problem is complicated by the fact that, in an optimal schedule, optional service times of a given task may *vary* from instance to instance which makes the framework of classical periodic scheduling theory inapplicable. Furthermore, this fact introduces a large number of variables in any analytical approach. Finally, by allowing nonlinear reward functions to better characterize the optional tasks' contribution to the overall system, the optimization problem becomes computationally harder.

In [5], Chung, Liu and Lin proposed the strategy of assigning *statically* higher priorities to mandatory parts. This decision, as proven in that paper, effectively achieves the first objective mentioned above by securing mandatory parts from the potential interference of optional parts. Optional parts are scheduled whenever no mandatory part is ready in the system. In [5], the simulation results regarding the performance of several policies which assign static or dynamic priorities among optional parts are reported. We call the class of algorithms that statically assign higher priorities to mandatory parts *Mandatory-First Algorithms*.

In our solution, we do *not* decouple the objectives of meeting the deadlines of mandatory parts and maximizing the total (or average) reward. We formulate the periodic reward-based scheduling problem as an optimization problem and derive an important and surprising property of the solution for the most common (i.e., linear and concave) reward functions. Namely, *we prove that there is always an optimal schedule where optional service times of a given task do not vary from instance to instance.* This important result immediately implies that the optimality (in terms of achievable utilization) of any policy which can fully use the processor in case of hard-real time periodic tasks also holds in the context of reward-based scheduling (in terms of total reward) *when used with these optimal service times*. Examples of such policies are RMS-h (RMS with harmonic periods) [14], EDF [14] and LLF [16] scheduling disciplines.

Following these existence proofs, we address the problem of efficiently computing optimal service times and provide polynomial-time algorithms for linear and/or general concave reward functions. Note that using these optimal and constant optimal service times has also important practical advantages: (a) The runtime overhead due to the existence of mandatory/optional dichotomy and reward functions is removed, and (b) existing RMS-h, EDF and LLF schedulers may be used without any modification with these optimal assignments.

The remainder of this paper is organized as follows: In Section 2, the system model and basic definitions are given. The main result about the optimality of any periodic policy which can fully utilize the processor(s) is obtained in Section 3. In Section 4, we first analyze the worst-case performance of Mandatory-First approaches. We also provide the results of experiments on a synthetic task set to compare the performance of policies proposed in [5] against our optimal algorithm. In Section 5, we show that the concavity assumption is also necessary for computational efficiency by proving that allowing convex reward functions results in an NP-Hard problem. We present details about the specific optimization problem that we use in Section 6. We conclude by summarizing our contribution and discussing future work.

2 System Model

We consider a set **T** of n periodic real-time tasks $T_1, T_2, \ldots, T_n$ on a uniprocessor system. The period of T_i is denoted by P_i, which is also equal to the deadline of the current invocation. We refer to the j^{th} invocation of task T_i as T_{ij}. All tasks are assumed to be independent and ready at $t = 0$.

Each task T_i consists of a mandatory part M_i and an optional part O_i. The length of the mandatory part is denoted by m_i; each task must receive at least m_i units of service time before its deadline in order to provide output of acceptable quality. The optional part O_i becomes ready for execution only when the mandatory part M_i completes.

Associated with each optional part of a task is a reward function $R_i(t_{ij})$ which indicates the reward accrued by task T_{ij} when it receives t_{ij} units of service *beyond its mandatory portion*. $R_i(t_{ij})$ is of the form:

$$R_i(t_{ij}) = \begin{cases} f_i(t_{ij}) & \text{if} \quad 0 \leq t_{ij} \leq o_i \\ f_i(o_i) & \text{if} \quad t_{ij} > o_i \end{cases} \tag{1}$$

where f_i is a nondecreasing, concave and differentiable function over nonnegative real numbers and o_i is the length of *entire* optional part O_i. We underline that $f_i(t_{ij})$ is nondecreasing: the reward accrued by task T_{ij} can not decrease by allowing it to run longer. Notice that the reward function $R_i(t)$ is not necessarily differentiable at $t = o_i$. Note also that in this formulation, by the time the task's optional execution time t reaches the threshold value o_i, the reward accrued ceases to increase.

A schedule of periodic tasks is **feasible** if mandatory parts meet their deadlines at every invocation. Given a feasible schedule of the task set **T**, the **average reward** of task T_i is:

$$REW_i = \frac{P_i}{P} \sum_{j=1}^{P/P_i} R_i(t_{ij}) \tag{2}$$

where P is the *hyperperiod*, that is, the least common multiple of $P_1, P_2, \ldots, P_n$ and t_{ij} is the service time assigned to the j^{th} instance of optional part of task T_i. That is, the average reward of T_i is computed over the number of its invocations during the hyperperiod P, in an analogous way to the definition of average error in [5].

The **average weighted reward** of a feasible schedule is then given by:

$$REW_W = \sum_{i=1}^{n} w_i \; REW_i \tag{3}$$

where w_i is a constant in the interval (0,1] indicating the relative importance of optional part O_i. Although this is the most general formulation, it is easy to see that the weight w_i can always be incorporated into the reward function $f_i()$, by replacing it by $w_i \, f_i()$. Thus, we will assume that all weight (importance) information are already expressed in the reward function formulation and that REW_W is simply equal to $\sum_{i=1}^{n} REW_i$.

Finally, a schedule is **optimal** if it is feasible and maximizes the average weighted reward.

<u>A Motivating Example:</u>

Before describing our solution to the problem, we present a simple example which shows the performance limitations of *any* Mandatory-First algorithm. Consider two tasks where $P_1 = 4, m_1 = 1, o_1 = 1, P_2 = 8, m_2 = 3, o_2 = 5$. Assume that the reward functions associated with optional parts are linear and $f_1(t_1) = k_1 t_1$, $f_2(t_2) = k_2 t_2$, where $k_1 \gg k_2$. In this case, the "best" algorithm among "Mandatory-First" approaches should produce the schedule shown in Figure 1.

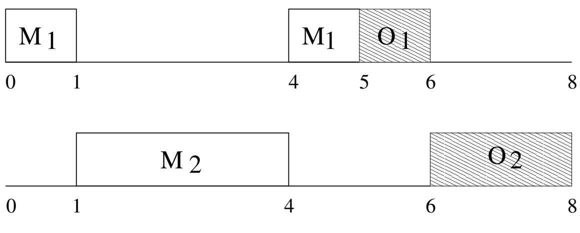


Figure 1. A schedule produced by Mandatory-First Algorithm

In Figure 1, the Rate Monotonic Priority Assignment is used whenever more than one mandatory task are simultaneously ready, as in [5]. Yet, following other (dynamic or static) priority schemes would not change the fact that the processor will be busy executing solely mandatory parts until $t = 5$ under any Mandatory-First approach. During the remaining idle interval [5,8], the best algorithm would have chosen to schedule

O_1 completely (which brings most benefit to the system) for 1 time unit and O_2 for 2 time units. However, an optimal algorithm would produce the schedule depicted in Figure 2.

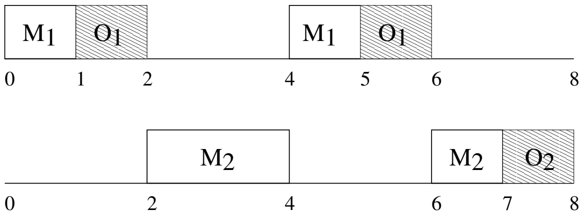


Figure 2. The optimal schedule

As it can be seen, the optimal strategy in this case consisted of delaying the execution of M_2 in order to be able to execute 'valuable' O_1 and we would still meet the deadlines of all mandatory parts. By doing so, we would succeed in executing two instances of O_1, in contrast to any Mandatory-First scheme which can execute only one instance of O_1. Remembering that $k_1 \gg k_2$, one can conclude that the reward accrued by the 'best' Mandatory-First scheme may only be around half of that accrued by the optimal one, for this example. Also, observe that in the optimal schedule, the optional execution times of a given task did not vary from instance to instance. In the next section, we prove that this pattern is not a mere coincidence. We further perform an analytical worst-case analysis of Mandatory-First algorithms in Section 4.

3 Optimality of Full-Utilization Policies for Periodic Reward-Based Scheduling

The objective of the Periodic Reward-Based Scheduling problem is clearly finding optimal $\{t_{ij}\}$ values to maximize the average reward. By substituting the average reward expression given by (2) in (3), we obtain our objective function:

$$\text{maximize} \quad \sum_{i=1}^{n} \frac{P_i}{P} \sum_{j=1}^{P/P_i} R_i(t_{ij})$$

The first constraint we must enforce is that the total processor demand of mandatory and optional parts during the hyperperiod P may not exceed the available computing capacity:

$$\sum_{i=1}^{n} \sum_{j=1}^{P/P_i} (m_i + t_{ij}) \leq P$$

Note that this constraint is necessary, but not sufficient for feasibility of the task set with $\{m_i\}$ and $\{t_{ij}\}$ values. Next, we observe that optimal t_{ij} values may not be less than zero, since negative service times do not have any physical interpretation. In addition, the service time of an optional instance of T_i does not need to exceed the upperbound o_i of reward function $R_i(t)$, since the reward accrued by T_i ceases to increase after $t_{ij} = o_i$. Hence, we obtain our second constraint set:

$$0 \leq t_{ij} \leq o_i \quad i = 1, \ldots, n \quad j = 1, \ldots, \frac{P}{P_i}$$

The above constraint allows us also to readily substitute $f_i()$ for $R_i()$ in the objective function. Finally, we need to express the "full" feasibility constraint, requiring that mandatory parts complete in a timely manner at every invocation. Note that it is sufficient to have one feasible schedule for task T_i with m_i and the involved optimal $\{t_{ij}\}$ values.

To re-capture all the constraints, the periodic reward-based scheduling problem, which we denote by REW-PER, is to find $\{t_{ij}\}$ values so as to:

$$\text{maximize} \quad \sum_{i=1}^{n} \frac{P_i}{P} \sum_{j=1}^{P/P_i} f_i(t_{ij}) \qquad (4)$$

subject to

$$\sum_{i=1}^{n} \frac{P}{P_i} m_i + \sum_{i=1}^{n} \sum_{j=1}^{P/P_i} t_{ij} \leq P \qquad (5)$$

$$0 \leq t_{ij} \leq o_i \quad i = 1, \ldots, n \quad j = 1, \ldots, \frac{P}{P_i} \qquad (6)$$

$$\text{A feasible schedule exists with } \{m_i\} \text{ and } \{t_{ij}\} \text{ values} \qquad (7)$$

We express this last constraint in English and not through formulas since the algorithm producing this schedule including optimal t_{ij} assignments need not be specified at this point.

Before stating our main result, we underline that if $\sum_{i=1}^{n} \frac{P}{P_i} m_i > P$, it is not possible to schedule mandatory parts in a timely manner and the optimization problem has no solution. Note that this condition is equivalent to $\sum_{i=1}^{n} \frac{m_i}{P_i} > 1$, which indicates that the task set would be unschedulable, even if it consisted of only mandatory parts. Hence, hereafter, we assume that $\sum_{i=1}^{n} \frac{m_i}{P_i} \leq 1$ and therefore there exists at least one feasible schedule.

Theorem 1 *Given an instance of Problem REW-PER, there exists an optimal solution where the optional parts of a task T_i receive the* same *service time at every instance, i.e. $t_{ij} = t_{ik}$ $1 \leq j < k \leq \frac{P}{P_i}$. Furthermore, any periodic hard-real time scheduling policy which can fully utilize the processor (EDF, LLF, RMS-h) can be used to obtain a feasible schedule with these assignments.*

Proof: Our strategy to prove the theorem will be as follows. We will drop the feasibility condition (7) and obtain a new optimization problem whose feasible region strictly contains that of REW-PER. Specifically, we consider a new optimization problem, denoted by MAX-REW, where the objective function is again given by (4), but only the constraint sets (5) and (6) have to be satisfied. Note that the new problem MAX-REW does *not* a priori correspond to any scheduling problem, since the feasibility issue is not addressed. We then show that there exists an optimal solution of MAX-REW where $t_{ij} = t_{ik}$ $1 \leq j < k \leq \frac{P}{P_i}$. Then, we will return to REW-PER and demonstrate the existence of a feasible schedule (i.e. satisfiability of (7)) under these assignments. The reward associated with MAX-REW's optimal solution is always greater than or equal to that of REW-PER's optimal solution, for MAX-REW does *not* consider one of the REW-PER's constraints. This will imply that this specific optimal solution of the new problem MAX-REW is also an optimal solution of REW-PER.

Now, we show that there exists an optimal solution of MAX-REW where $t_{ij} = t_{ik}$ $1 \leq j < k \leq \frac{P}{P_i}$.

Claim 1 *Let $\{t_{ij}\}$ be an optimal solution to MAX-REW, $1 \leq i \leq n \quad 1 \leq j \leq \frac{P}{P_i} = q_i$. Then $\{t'_{ij}\}$ where $t'_{i1} = t'_{i2} = \ldots = t'_{iq_i} = t'_i = \frac{t_{i1}+t_{i2}+\ldots+t_{iq_i}}{q_i} \quad 1 \leq i \leq n \quad 1 \leq j \leq q_i$, is also an optimal solution to MAX-REW.*

- We first show that $\{t'_{ij}\}$ values satisfy the constraints (5) and (6), if $\{t_{ij}\}$ already satisfy them. Since $\sum_{j=1}^{q_i} t_{ij} = \sum_{j=1}^{q_i} t'_{ij} = q_i t'_i$ the constraint (5) is not violated by the transformation. Also, by assumption, $t_{ij} \leq o_i \, \forall j$, which implies $\max_j \{t_{ij}\} \leq o_i$. Since t'_i, which is arithmetic mean of $t_{i1}, t_{i2}, \ldots, t_{iq_i}$ is necessarily less than or equal to $\max_j \{t_{ij}\}$, the constraint set (6) is not violated either by the transformation.
- Furthermore, the total reward does not decrease by this transformation, since $\sum_{j=1}^{q_i} f_i(t_{ij}) \leq q_i \; f_i(t'_i)$. The proof of this statement is presented in the Appendix.

Using Claim 1, we can commit to finding *an* optimal solution of MAX-REW by setting $t_{i1} = t_{i2} = \ldots = t_{iq_i} = t_i \quad i = 1, \ldots, n$. In this case, $\sum_{j=1}^{P/P_i} f_i(t_{ij}) = \frac{P}{P_i} f_i(t_i)$ and $\sum_{j=1}^{P/P_i} t_{ij} = \frac{P}{P_i} t_i$. Hence, this version of MAX-REW can be re-written as:

$$\text{maximize} \quad \sum_{i=1}^{n} f_i(t_i) \tag{8}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{P}{P_i} t_i \leq P - \sum_{i=1}^{n} \frac{P}{P_i} m_i \tag{9}$$

$$0 \leq t_i \leq o_i \quad i = 1, \ldots, n \tag{10}$$

Finally, we prove that the optimal solution $t_1, t_2, \ldots, t_n$ of MAX-REW above, automatically satisfies the feasibility constraint (7) of our original problem REW-PER. Having equal optional service times for a given task greatly simplifies the verification of this constraint. Since $t_1, t_2, \ldots, t_n$ (by assumption) satisfy (9), we can write $\sum_{i=1}^{n} P \cdot \frac{m_i + t_i}{P_i} \leq P$, or equivalently, $\sum_{i=1}^{n} \frac{m_i + t_i}{P_i} \leq 1$.

This implies that any policy which can achieve 100% processor utilization in classical periodic scheduling theory (EDF, LLF, RMS-h) can be used to obtain a feasible schedule for tasks, which have now identical execution times $m_i + t_i$ at every instance. Hence, the "full feasibility" constraint (7) of REW-PER is satisfied. Furthermore, this schedule clearly maximizes the average reward since $\{t_i\}$ values maximize MAX-REW whose feasible region encompasses that of REW-PER.

□

Corollary 1 *Optimal t_i values for the Problem REW-PER can be found by solving the optimization problem given by (8), (9) and (10).*

We discuss the solution of this concave optimization problem in Section 6.

4 Evaluation and comparison with other approaches

We showed through the example in Section 2 that the reward accrued by *any* Mandatory-First scheme [5] may only be approximately half of that of the optimal algorithm. We now prove that, under the worst-case scenario, the ratio of the reward accrued by a Mandatory-First approach to the reward of the optimal algorithm approaches zero.

Theorem 2 *There is an instance of the periodic reward-based scheduling problem where, for any integer $r \geq 2$, the ratio* $\frac{\text{Reward of the best mandatory-first scheme}}{\text{Reward of the optimal scheme}} = \frac{2}{r}$

Proof: Consider two tasks T_1 and T_2 such that $P_2/P_1 = r$, $f_1(t_1) = k_1 t_1$, $f_2(t_2) = k_2 t_2$ and $k_1/k_2 = r(r-1)$. Furthermore, let $m_2 = \frac{1}{2}(r\, o_2)$ and

$$P_1 = m_1 + o_1 + \frac{m_2}{r} = m_1 + \frac{m_2}{r-1}$$

which implies that $o_1 = \frac{m_2}{r\,(r-1)}$.

This setting suggests that during any period of T_1, a scheduler has the choice of executing (parts of) O_1 and/or M_2, in addition to M_1.

Note that under any Mandatory-First policy, the processor will be continuously busy executing mandatory parts until $t = P_2 - P_1 + m_1$. Furthermore, the best algorithm among Mandatory-First policies should use the remaining idle times by scheduling O_1 entirely (since $k_1 > k_2$) and $t_2 = \frac{m_2}{r} = \frac{o_2}{2}$ units of O_2. The resulting schedule is shown in Figure 3.

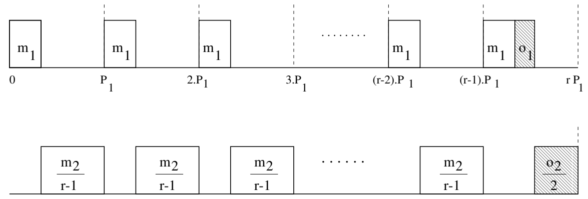


Figure 3. A schedule produced by Mandatory-First Algorithm

The average reward that the best mandatory-first algorithm (MFA) can accrue is therefore:

$$R_{MFA} = \frac{f_1(o_1)}{r} + f_2(t_2)$$

However, an optimal algorithm (shown in Figure 4) would choose delaying the execution of M_2 for o_1 units of time, at

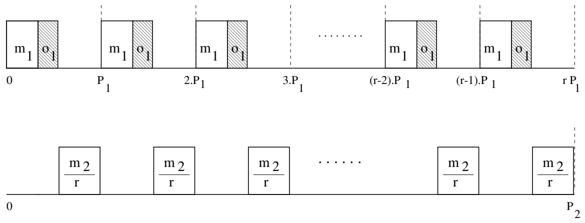


Figure 4. An optimal schedule

every period of T_1. By doing so, it would have the opportunity of accruing the reward of O_1 at every instance.

The total reward of the optimal schedule is:

$$R_{OPT} = \frac{r\, f_1(o_1)}{r} = f_1(o_1)$$

The ratio of rewards for the two policies turns out to be (for any $r \geq 2$)

$$\frac{R_{MFA}}{R_{OPT}} = \frac{1}{r} + \frac{f_2(t_2)}{f_1(o_1)} = \frac{1}{r} + \frac{1}{r(r-1)}\frac{m_2}{r}\frac{r(r-1)}{m_2} = \frac{2}{r}$$

which can be made as close as possible to 0 by appropriately choosing r (i.e., choosing a large value for r).

□

Theorem 2 gives the worst-case performance ratio of Mandatory-First schemes. We also performed experiments with a synthetic task set to investigate the relative performance of Mandatory-First schemes proposed in [5] with different types of reward functions and different mandatory/optional workload ratios.

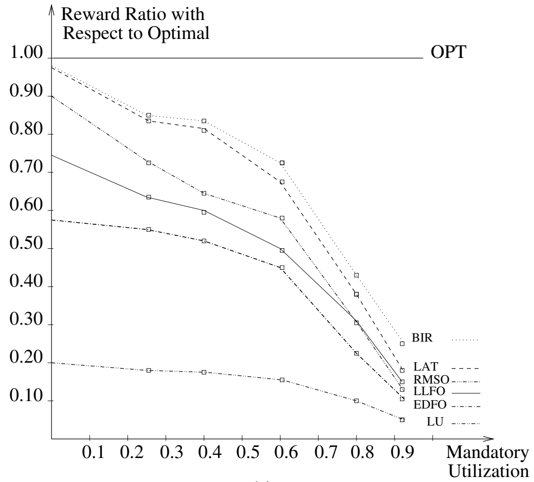
The Mandatory-First schemes differ by the policy according to which optional parts are scheduled when there is no mandatory part ready to execute. *Rate-Monotonic (RMSO)* and *Least-Utilization (LU)* schemes assign statically higher priorities to *optional parts* with smaller periods and least utilizations respectively. Among dynamic priority schemes are *Earliest-Deadline-First (EDFO)* and *Least-Laxity-First (LLFO)* which consider the deadline and laxity of optional parts when assigning priorities. *Least Attained Time (LAT)* aims at balancing execution times of optional parts that are ready, by dispatching the one that executed *least* so far. Finally, *Best Incremental Return (BIR)* is an on-line policy which chooses the optional task contributing most to the total reward, at a given *slot*. In other words, *at every slot* BIR selects the optional part O_{ij} such that the difference $f_i(t_{ij} + \Delta) - f_i(t_{ij})$ is the largest (here t_{ij} is the optional service time O_{ij} has already received and Δ is the minimum time slot that the scheduler assigns to any optional task). However, it is still a sub-optimal policy since it does not consider the laxity information. The authors indicate in [5] that BIR is too computationally complex to be actually implemented. However, since the total reward accrued by BIR is usually much higher than the other five policies, BIR is used as a yardstick for measuring the performance of other algorithms.

We have used a synthetic task set comprising 11 tasks whose total (mandatory + optional) utilization is 2.3. Individual task utilizations vary from 0.03 to 0.6. Considering exponential, logarithmic and linear reward functions as separate cases, we compared the reward of six Mandatory-First schemes with our optimal algorithm (OPT). The tasks' characteristics (including reward functions) are given in the Table below. In our experiments, we first set mandatory utilization to 0 (which corresponds to the case of all-optional workload), then increased it to 0.25, 0.4, 0.6, 0.8 and 0.91 subsequently.

Task	P_i	$m_i + o_i$	$f_i^1(t)$	$f_i^2(t)$	$f_i^3(t)$
T_1	20	10	$15\,(1-e^{-t})$	$7\ \ln(20\,t+1)$	$5\,t$
T_2	30	18	$20\,(1-e^{-3t})$	$10\ \ln(50\,t+1)$	$7\,t$
T_3	40	5	$4\,(1-e^{-t})$	$2\ \ln(10\,t+1)$	$2\,t$
T_4	60	2	$10\,(1-e^{-0.5t})$	$5\ \ln(5\,t+1)$	$4\,t$
T_5	60	2	$10\,(1-e^{-0.2t})$	$5\ \ln(25\,t+1)$	$4\,t$
T_6	80	12	$5\,(1-e^{-t})$	$3\ \ln(30\,t+1)$	$2\,t$
T_7	90	18	$17\,(1-e^{-t})$	$8\ \ln(8\,t+1)$	$6\,t$
T_8	120	15	$8\,(1-e^{-t})$	$4\ \ln(6\,t+1)$	$3\,t$
T_9	240	28	$8\,(1-e^{-t})$	$4\ \ln(9\,t+1)$	$3\,t$
T_{10}	270	60	$12\,(1-e^{-0.5t})$	$6\ \ln(12\,t+1)$	$5\,t$
T_{11}	2160	300	$5\,(1-e^{-t})$	$3\ \ln(15\,t+1)$	$2\,t$

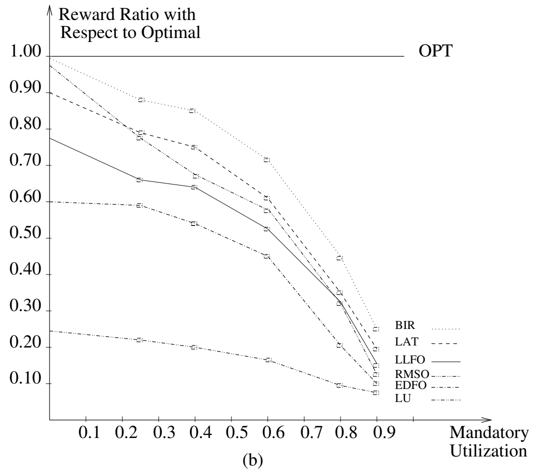
In our experiments, a common pattern appears: the optimal algorithm improves more dramatically with the increase in mandatory utilization. The other schemes miss the opportunities of executing "valuable" optional parts while constantly favoring mandatory parts. The reward loss becomes striking as the mandatory workload increases. Figures 5 and 6 show the reward ratio for the case of exponential and logarithmic reward functions, respectively. The curves for these strictly concave reward functions are fairly similar: BIR performs best among Mandatory-First schemes, and its performance degrades as the mandatory utilization increases; for instance the ratio falls to 0.73 when mandatory utilization is 0.6. Other algorithms which are more amenable to practical implementations (in terms of runtime overhead) than BIR perform even worse. However, it is worth noting that the performance of LAT is close to that of BIR. This is to be expected, since task sets with strictly concave reward functions usually benefit from "balanced" optional service times.

Figure 7 shows the reward ratio for linear reward functions. Although the reward ratio of Mandatory-First schemes again decreases with the mandatory utilization, the decrease is less dramatic than in the case of concave functions. However, note that the ratio is typically less than 0.5 for the five practical schemes. It is interesting to observe that the (impractical) BIR's reward now remains comparable to that of optimal, even in the higher mandatory utilizations: the difference is less than 15%. The main reason for this behavior change lies on the fact that, for a given task, the reward of optional execution slots in different instances does not make a difference in the linear case. In contrast, not executing the "valuable" *first slot(s)* of a given instance creates a tremendous effect for nonlinear



(a)

Figure 5. The Reward Ratio of Mandatory-First schemes: exponential reward functions



(b)

Figure 6. The Reward Ratio of Mandatory-First schemes: logarithmic reward functions

concave functions. The improvement of the optimal algorithm would be larger for a larger range of k_i values (where k_i is the coefficient of the linear reward function). We recall that the worst-case performance of BIR may be arbitrarily bad with respect to the optimal one for linear functions, as Theorem 2 suggests.

5 Periodic Reward-Based Scheduling Problem with Convex Reward Functions is NP-Hard

As we mentioned before, maximizing the total (or average) reward with 0/1 constraints case had already been proven to be NP-Complete in [15]. In this section, we show that convex reward functions also result in an NP-Hard problem.

We now show how to transform the SUBSET-SUM problem, which is known to be NP-Complete, to REW-PER with convex reward functions.

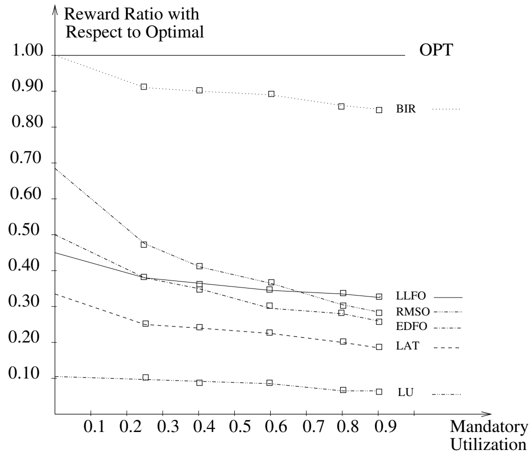


Figure 7. The Reward Ratio of Mandatory-First schemes: linear reward functions

SUBSET-SUM: Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of positive integers and the integer M, is there a set $S_A \subseteq S$ such that $\sum_{s_i \in S_A} s_i = M$?

We construct the corresponding REW-PER instance as follows. Let $W = \sum_{i=1}^{n} s_i$. Now consider a set of n periodic tasks with the same period M and mandatory parts $m_i = 0 \ \forall i$. The reward function associated with T_i is given by:

$$R_i(t_i) = \begin{cases} f_i(t_i) & \text{if} \quad 0 \leq t \leq o_i = s_i \\ f_i(o_i) & \text{if} \quad t > o_i = s_i \end{cases}$$

where $f_i(t_i) = t_i^2 + (W - s_i)t_i$ is a strictly convex and increasing function on nonnegative real numbers.

Notice that $f_i(t_i)$ can be re-written as $t_i(t_i - s_i) + W\,t_i$. Also we underline that having the same periods for all tasks implies that REW-PER can be formulated as:

$$\text{maximize} \qquad \sum_{i=1}^{n} t_i\,(t_i - s_i) + W \sum_{i=1}^{n} t_i \tag{11}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} t_i \leq M \tag{12}$$

$$0 \leq t_i \leq s_i \tag{13}$$

Let us denote by $MaxRew$ the total reward of the optimal schedule. Observe that for $0 < t_i < s_i$, the quantity $t_i\,(t_i - s_i) < 0$. Otherwise, at either of the boundary values 0 or s_i, $t_i\,(t_i - s_i) = 0$. Hence, $MaxRew \leq WM$.

Now, consider the question: "Is $MaxRew$ equal to WM?". Clearly, this question can be answered quickly if there is a polynomial-time algorithm for REW-PER where reward functions are allowed to be convex. Furthermore, the answer can be positive only when $\sum_{i=1}^{n} t_i = M$ *and* each t_i is equal to either 0 or s_i. Therefore, $MaxRew$ equal to WM, if and only if there is a set $S_A \subseteq S$ such that $\sum_{s_i \in S_A} s_i = M$, which implies that REW-PER with convex reward functions is NP-Hard.

6 Solution of Periodic Reward-Based Scheduling Problem with Concave Reward Functions

Corollary 1 reveals that the optimization problem whose solution provides optimal service times is of the form:

$$\begin{array}{ll} \text{maximize} & \sum\limits_{i=1}^{n} f_i(t_i) \\ \text{subject to} & \sum\limits_{i=1}^{n} b_i\ t_i \leq d \\ & t_i \leq o_i \quad i = 1, 2, ...n \\ & 0 \leq t_i \quad i = 1, 2, ...n \end{array}$$

where d (the 'slack' available for optional parts) and $b_1, b_2, \ldots, b_n$ are positive rational numbers. In this section, we present polynomial-time solutions for this problem, where each f_i is a nondecreasing, concave and differentiable[1] function.

First note that, if the available slack is large enough to accommodate every optional part entirely (i.e., if $\sum_{i=1}^{n} b_i\ o_i \leq d$), then the choice $t_i = o_i \ \forall\, i$ clearly maximizes the objective function due to the nondecreasing nature of reward functions.

Otherwise, the slack d should be used in its entirety since the total reward never decreases by doing so (again due to the nondecreasing nature of the reward functions). In this case, we obtain a concave optimization problem with lower and upper bounds, denoted by OPT-LU. An instance of OPT-LU is specified by the set of nondecreasing concave functions $\mathbf{F} = \{f_1, \ldots, f_n\}$, the set of upper bounds $\mathbf{O} = \{o_1, \ldots, o_n\}$ and the available slack d. The aim is to:

$$\text{maximize} \quad \sum_{i=1}^{n} f_i(t_i) \tag{14}$$

$$\text{subject to} \quad \sum_{i=1}^{n} b_i\ t_i = d \tag{15}$$

$$t_i \leq o_i \quad i = 1, 2, ...n \tag{16}$$

$$0 \leq t_i \quad i = 1, 2, ...n \tag{17}$$

where $0 < d < \sum_{i=1}^{n} b_i \cdot o_i$.

Special Case of Linear functions: If **F** comprises solely linear functions, the solution can be considerably simplified. Note that for a function $f_i(t_i) = k_i \cdot t_i$, if we increase t_i by Δ then total reward increases by $k_i\,\Delta$. However by doing so, we make use of $b_i\,\Delta$ units of slack (d is reduced by $b_i\,\Delta$ due to (15)). Hence, the "marginal return" of task T_i per slack unit is $w_i = \frac{k_i}{b_i}$. It is therefore possible to order the functions according to their marginal return, and distribute the slack in decreasing order of marginal returns, while taking account the upper bounds. We note that this solution is analogous to the one presented in [17]. The dominant factor in the time complexity comes from the initial sorting procedure, hence in the special case of all-linear functions, OPT-LU can be solved in time $O(n \log n)$.

[1] In the auxiliary optimization problems which will be introduced shortly, the differentiability assumption holds as well.

When **F** contains nonlinear functions then the procedure becomes more involved. In the next two subsections, we introduce two auxiliary optimization problems, namely Problem OPT (which considers only the equality constraint) and Problem OPT-L (which considers only the equality and lower bound constraints), which will be used to solve OPT-LU.

6.1 Problem OPT: Equality Constraints Only

The problem OPT is characterized by:

$$\begin{array}{ll} \text{maximize} & \sum\limits_{i=1}^{n} f_i(t_i) \\ \text{subject to} & \sum\limits_{i=1}^{n} b_i\ t_i = d \end{array}$$

where $\mathbf{F} = \{f_1, \ldots, f_n\}$ is the set of nondecreasing concave functions, possibly including some linear function(s). As it can be seen, OPT does not consider the lower and upper bound constraints of Problem OPT-LU. The algorithm which returns the solution of Problem OPT, is denoted by "Algorithm OPT".

When **F** is composed solely of *non-linear* reward functions, the application of Lagrange multipliers technique to the Problem OPT, yields:

$$\frac{1}{b_i} f_i'(t_i) - \lambda = 0 \qquad i = 1, 2, \ldots, n \tag{18}$$

where λ is the common Lagrange multiplier and $f_i'(t_i)$ is the derivative of the reward function f_i. The quantity $\frac{1}{b_i} f_i'(t_i)$ in (18) actually represents the *marginal return* contributed by T_i to the total reward, which we will denote as $w_i(t_i)$. Observe that since $f_i(t_i)$ is non-decreasing and concave by assumption, both $w_i(t_i)$ and $f_i'(t_i)$ is non-increasing and positive valued. Equation (18) implies that the marginal returns $w_i(t_i) = \frac{1}{b_i} f_i'(t_i)$ *should* be equal for all reward functions in the optimal solution $\{t_1, \ldots, t_n\}$. Considering that the equality $\sum\limits_{i=1}^{n} b_i\ t_i = d$ should also hold, one can obtain closed formulas in most of the cases which occur in practice. The closed formulas presented below are obtained by this method.

- For *logarithmic* reward functions of the form $f_i(t_i) = \ln(k_i \cdot t_i + c_i)$,
  $$t_1 = \frac{d + \sum\limits_{j=1}^{n} \frac{c_j}{k_j} - \frac{c_1}{k_1\,b_1} \sum\limits_{j=1}^{n} b_j}{\frac{1}{b_1} \sum\limits_{j=1}^{n} b_j}$$
  $$t_j = b_1\,t_1 + \frac{c_1}{k_1\,b_1} - \frac{c_j}{k_j\,b_j} \quad \forall j \;\; 1 < j \leq n.$$
- For *exponential* reward functions of the form $f_i(t_i) = c_i(1 - e^{-k_i t_i})$,
  $$t_1 = \frac{d - \sum\limits_{j=1}^{n} \frac{1}{k_j} [\ln(\frac{c_j b_1 k_j}{c_1 b_j k_1})]}{\sum\limits_{j=1}^{n} \frac{k_1}{k_j}}$$
  $$t_j = \frac{1}{k_j}[k_1\,t_1 + \ln(\frac{c_j b_1 k_j}{c_1 b_j k_1})] \quad \forall j \;\; 1 < j \leq n.$$

- For "k^{th} root" reward functions of the form
$f_i(t_i) = c_i\ t_i^{1/k}\ (k > 1)$,
$t_1 = \frac{d}{\sum\limits_{j=1}^{n} (\frac{b_j c_1}{b_1 c_j})^{\frac{1}{k-1}}}$
$t_j = t_1\, (\frac{b_j c_1}{b_1 c_j})^{\frac{1}{k-1}} \quad \forall j\ \ 1 < j \leq n.$

When it is not possible to find a closed formula, following exactly the approach presented in [10, 13], we solve λ in the equation $\sum_{i=1}^{n} b_i\ \ h_i(\lambda) = d$, where $h_i(k)$ is the inverse function of $\frac{1}{b_i} f_i'(t_i) = w_i(t_i)$ (we assume the existence of the derivative's inverse function whenever f_i is nonlinear, complying with [10, 13]). Once λ is determined, $t_i = h_i(\lambda),\ i = 1, \ldots, n$ is the optimal solution.

We now examine the case where **F** is a *mix* of linear and nonlinear functions. Consider two linear functions $f_i(t) = k_i \cdot t$ and $f_j(t) = k_j \cdot t$. The marginal return of $f_i(t_i)$ is $w_i(t_i) = \frac{k_i}{b_i} = w_i$ and that of f_j is $w_j(t_j) = \frac{k_j}{b_j} = w_j$. I $w_j > w_i$ then the service time t_i should be definitely zero, since marginal return of f_i is strictly less than f_j everywhere. After this elimination process, if there are $p > 1$ linear functions with the same (largest) marginal return w_{max} then we will consider them as a single linear function in the procedure below and evenly divide the returned service time t_{max} among t_j values corresponding to these p functions.

Hence, without loss of generality, we assume that $f_n(t) = k_n \cdot t$ is the only linear function in **F**. Its marginal return is $w_n(t_n) = \frac{k_n}{b_n} = w_{max}$. We first compute the optimal distribution of slack d among tasks with nonlinear reward functions $f_1, \ldots, f_{n-1}$. By the Lagrange multipliers technique, $w_i(t_i) - \lambda = 0$ and thus $w_1(t_1^*) = w_2(t_2^*) = \ldots = w_{n-1}(t_{n-1}^*) = \lambda$ at the optimal solution $t_1^*, t_2^*, \ldots, t_{n-1}^*$.

Now we distinguish two cases:

- $\lambda \geq max$. In this case, $t_1^*, t_2^*, \ldots, t_{n-1}^*$ and $t_n = 0$ is the optimal solution to OPT. To see this, first remember that all the reward functions are concave and nondecreasing, hence $w_i(t_i^* - \epsilon) \geq w_i(t_i^*) \geq w_n(\epsilon) = w_{max}\ i = 1, \ldots, n-1$ for all $\epsilon >= 0$. This implies that transferring some service time from another task T_i to T_n would mean favoring the task which has the smaller marginal reward rate and would not be optimal.

- $\lambda < w_{max}$. In this case, reserving the slack d solely to tasks with nonlinear reward functions means violating the best marginal rate principle and hence is not optimal. Therefore, we should increase service time t_i until $w_i(t_i)$ drops to the level of w_{max} for $i = 1, 2, \ldots, n-1$, *but not beyond*. Solving $h_i(w_{max}) = t_i$ for $i = 1, 2, \ldots, n-1$ and assigning any remaining slack $\frac{d - \sum_{i=1}^{n-1} t_i}{b_n}$ to t_n (the service time of unique task with linear reward function) clearly satisfies the best marginal rate principle and achieves optimality.

6.2 Problem OPT-L: Equality Constraints with Lower Bounds

Now, we consider the optimization problem with the equality and lower bound constraints, denoted by OPT-L. An instance of Problem OPT-L is characterized by the set **F**=$\{f_1, f_2, .., f_n\}$ of nondecreasing concave reward functions, and the available slack d:

$$\text{maximize} \qquad \sum_{i=1}^{n} f_i(t_i) \tag{19}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} b_i\ t_i = d \tag{20}$$

$$0 \leq t_i \quad i = 1, 2, \ldots n \tag{21}$$

To solve OPT-L, we first evaluate the solution set S_{OPT} to the corresponding problem OPT and check whether all inequality constraints are automatically satisfied. If this is the case, the solution set S_{OPT-L} of Problem OPT-L is clearly the solution S_{OPT}. Otherwise, we will construct S_{OPT-L} iteratively as described below.

Let $\Pi = \{x | \frac{1}{b_x} f_x'(0) \leq \frac{1}{b_i} f_i'(0)\ \ \forall i\}$. Remember that $\frac{1}{b_x} f_x'(t_x)$ is the marginal return associated with $f_x(t_x)$ and was denoted by $w_x(t_x)$. Informally, Π contains the functions[2] $f_x \in$ **F** with the smallest marginal returns at lower bound 0, $w_x(0)$.

Lemma 1 *If S_{OPT} violates some lower bound constraints of Problem OPT-L, then, in the optimal solution $t_m = 0\ \forall m \in \Pi$.*

The proof of Lemma 1 is based on Kuhn-Tucker optimality conditions for nonlinear optimization problems and is not presented here for lack of space (the complete proof can be found in [1]). The time complexity $C_{OPT}(n)$ of Algorithm OPT is $O(n)$ (If the mentioned closed formulas apply, then the complexity is clearly linear. Otherwise the unique unknown λ can be solved in linear time under concavity assumptions, as indicated in [10, 13]). Lemma 1 immediately implies the existence of an algorithm which sets $t_m = 0\ \forall m \in \Pi$, and then reinvokes Algorithm OPT for the remaining tasks and slack (in case that some inequality constraints are violated by S_{OPT}). Since the number of invocations is bounded by n, the complexity of the algorithm which solves OPT-L is $O(n^2)$.

Furthermore, it is possible to converge to the solution in time $O(n \log n)$ by using a binary-search like technique on Lagrange multipliers. Again, full details and correctness proof of this faster approach can be found in [1].

6.3 Problem OPT-LU: Equality Constraints with Upper and Lower Bounds

An instance of Problem OPT-LU is characterized by the set **F**= $\{f_1, f_2, \ldots, f_n\}$ of nondecreasing, differentiable, and

[2] We use the expression "functions" instead of "**indices** of functions" unless confusion arises.

concave reward functions, the set **O**= $\{o_1, o_2, .., o_n\}$ of upper bounds on optional parts, and available slack d:

$$\text{maximize} \qquad \sum_{i=1}^{n} f_i(t_i) \tag{22}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} b_i\, t_i = d \tag{23}$$

$$t_i \leq o_i \quad i = 1, 2, ...n \tag{24}$$

$$0 \leq t_i \quad i = 1, 2, ...n \tag{25}$$

We first observe the close relationship between the problems OPT-LU and OPT-L. Indeed, OPT-LU has only an additional set of upper bound constraints. It is not difficult to see that if S_{OPT-L} satisfies the constraints given by Equation (24), then the solution S_{OPT-LU} of problem OPT-LU is the same as S_{OPT-L}. However, if an upper bound constraint is violated then we will construct the solution iteratively in a way analogous to that described in the solution of Problem OPT-L.

Let $\Gamma = \{x | \frac{1}{b_x} f'_x(o_x) \geq \frac{1}{b_i} f'_i(o_i)\ \forall i\}$. Informally, Γ contains the functions $f_x \in \mathbf{F}$ with the largest marginal returns at the upper bounds, $w_x(o_x)$.

Lemma 2 *If S_{OPT-L} violates some lower bound constraints of Problem OPT-LU, then, in the optimal solution $t_m = o_m\ \forall m \in \Gamma$.*

The proof of Lemma 2 is again based on Kuhn-Tucker optimality conditions and can be found in [1].

The algorithm ALG-OPT-LU (see Figure 8) which finds solution to the problem OPT-LU is based on successively solving instances of OPT-L. First, we find the solution S_{OPT-L} of the corresponding problem OPT-L. We know that this solution is optimal for the simpler problem which does not take into account upper bounds. If the upper bound constraints are automatically satisfied, then we are done. However, if this is not the case, we set $t_q = o_q\ \forall q \in \Gamma$. Finally, we update the sets **F, O** and the slack d to reflect the fact that the values of $t_m\ \forall m \in \Gamma$ have been fixed.

Algorithm OPT-LU(F,O,d)

1 Set $S_{OPT-LU} = \emptyset$
2 if $\mathbf{F} = \emptyset$ then exit
3 Evaluate S_{OPT-L} /* consider only lower bounds */
4 if all upper bound constraints are satisfied then
$S_{OPT-LU} = (S_{OPT-LU} \cup S_{OPT-L})$; exit
5 compute $\mathbf{\Gamma}$
6 set $t_q = o_q \forall q \in \mathbf{\Gamma}$ in S_{OPT-LU}
7 set $d = d - \sum_{x \in \Gamma} b_x\ o_x$
8 set **F=F−Γ**
9 set **O=O**$-\{\mathbf{o_x} | \mathbf{x} \in \mathbf{\Gamma}\}$
10 Goto Step 2

Figure 8. Algorithm to solve Problem OPT-LU

Complexity: Notice that the worst case time complexity of each iteration is equal to that of Algorithm OPT-L, which is $O(n \cdot \log n)$. Furthermore, the cardinality of **F** decreases by at least 1 after each iteration. Hence, the number of iterations is bounded by n. It follows that the total time complexity of Algorithm OPT-LU is $O(n^2 \cdot \log n)$. However, in case of all-linear functions, OPT-LU can be solved in time $O(n \cdot \log n)$ as shown before.

7 Conclusion

In this paper, we have addressed the periodic reward-based scheduling problem in the context of uniprocessor systems. We proved that when the reward functions are convex, the problem is NP-Hard. Thus, our focus was on linear and concave reward functions, which adequately represent realistic applications such as image and speech processing, time-dependent planning and multimedia presentations. We have shown that there exists always a schedule where the optional execution times of a given task do not change from instance to instance. This result, in turn, implied the optimality of any periodic real-time policy which can achieve 100% utilization of the processor. The existence of such policies is well-known in real-time systems community: RMS-h, EDF and LLF. We have also presented polynomial-time algorithms for computing the optimal service times.

We underline that besides clear and observable reward improvement over previously proposed sub-optimal policies, our approach has the advantage of not requiring any runtime overhead for maximizing the reward of the system and for constantly monitoring the timeliness of mandatory parts. Once optimal optional service times are determined statically by our algorithm, an existing (e.g., EDF) scheduler does not need to be modified or to be aware of mandatory/optional semantic distinction at all. This appears as another major benefit of having pre-computed and optimal *equal* service times for a given task's invocations in reward-based scheduling.

In addition, Theorem 1 implies that as long as we are concerned with linear and concave reward functions, the resource allocation can be also made in terms of *utilization* of tasks without sacrificing optimality. In our opinion, this fact points to an interesting convergence of *instance-based* [5, 15] and *utilization-based* [17] models for the most common reward functions.

About the tractability issues regarding the nature of reward functions, the case of step functions was already proven to be NP-Complete ([15]). By solving efficiently the case of concave and linear reward functions and proving that the case of convex reward functions is NP-Hard, we believe that efficient solvability boundaries in (periodic or aperiodic) reward-based scheduling problem have been reached by our work in this aspect (assuming P $\neq$ NP).

We believe that considering dynamic aperiodic task arrivals, fault tolerance issues and investigating good approximation algorithms for intractable cases such as step functions and error cumulative jobs can be major avenues for reward-based scheduling.

Acknowledgements: The authors would like to thank the anonymous reviewers whose suggestions helped to improve the paper.

References

[1] H. Aydın, R. Melhem and D. Mossé. A Polynomial-time Algorithm to solve Reward-Based Scheduling Problem. *Technical Report 99-10*, Department of Computer Science, University of Pittsburgh, April 1999.

[2] G. Bernat and Alan Burns. Combining (n,m) Hard deadlines and Dual Priority scheduling. In *Proceedings of 18th IEEE Real-Time Systems Symposium*, December 1997.

[3] M. Boddy and T. Dean. Solving time-dependent planning problems. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* IJCAI-89, Aug 1989.

[4] E. Chang and A. Zakhor. Scalable Video Coding using 3-D Subband Velocity Coding and Multi-Rate Quantization. In *IEEE Int. Conf. on Acoustics, Speech and Signal processing*, July 1993.

[5] J.-Y. Chung, J. W.-S. Liu and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers,* 19(9): 1156-1173, September 1990.

[6] W. Feng and J. W.-S. Liu. An extended imprecise computation model for time-constrained speech processing and generation. In *Proceedings of the IEEE Workshop on Real-Time Applications*, May 1993.

[7] J. Grass and S. Zilberstein. Value-Driven Information Gathering. *AAAI Workshop on Building Resource-Bounded Reasoning Systems,* Rhode Island, 1997.

[8] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)- firm deadlines. *IEEE Transactions on Computers,* 44(12): 1443-1451, Dec 1995.

[9] E.J. Horvitz. Reasoning under varying and uncertain resource constraints *Proceedings of the Seventh National Conference on Artificial Intelligence,* AAAI-88, pp. 111-116, August 1988.

[10] J. K. Dey, J. Kurose and D. Towsley. On-Line Scheduling Policies for a class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks. *IEEE Transactions on Computers* 45(7):802-813, July 1996.

[11] G. Koren and D. Shasha. Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips. In *Proceedings of 16th IEEE Real-Time Systems Symposium*, December 1995.

[12] R. E. Korf. Real-time heuristic search. *Artificial Intelligence,* 42(2): pp.189 -212, 1990.

[13] C. M. Krishna and K. G. Shin. *Real-time Systems.* Mc Graw-Hill, New York 1997.

[14] C.L. Liu and J.W.Layland. Scheduling Algorithms for Multiprogramming in Hard Real-time Environment. *Journal of ACM* 20(1), 1973.

[15] 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. *IEEE Computer,* 24(5): 58-68, May 1991.

[16] A.K. Mok, *Fundamental Design Problems of Distributed systems for the Hard Real-Time Environment.* Ph.D. Dissertation, MIT, 1983.

[17] R. Rajkumar, C. Lee, J. P. Lehozcky and D. P. Siewiorek. A Resource Allocation Model for QoS Management. In *Proceedings of 18th IEEE Real-Time Systems Symposium*, December 1997.

[18] W.-K. Shih, J. W.-S. Liu, and J.-Y. Chung. Algorithms for scheduling imprecise computations to minimize total error. *SIAM Journal on Computing,* 20(3), July 1991.

[19] C. J. Turner and L. L. Peterson. Image Transfer: An end-to-end design. In *SIGCOMM Symposium on Communications Architectures and Protocols*, August 1992.

[20] S. V. Vrbsky and J. W. S. Liu. Producing monotonically improving approximate answers to relational algebra queries. In *Proceedings of IEEE Workshop on Imprecise and Approximate Computation,* December 1992.

[21] S. Zilberstein and S.J. Russell. Anytime Sensing, Planning and Action: A practical model for Robot Control. In *IJCAI 13*, 1993.

APPENDIX

We will show that:

$$\sum_{j=1}^{q_i} f_i(t_{ij}) \leq q_i \; f_i(t'_i) \tag{26}$$

where $t'_i = \frac{t_{i1}+t_{i2}+\ldots+t_{iq_i}}{q_i}$ and the function f_i is concave. If f_i is a linear function of the form $f_i(t) = k_i \cdot t$, then $\sum_{j=1}^{q_i} f_i(t_{ij}) = k_i\,(t_{i1} + t_{i2} + \ldots + t_{iq_i}) = k_i\, q_i\, t'_i$ and the inequality (26) is immediately established.

If f_i is general concave, we recall that:

$$\alpha f_i(x) + (1-\alpha) f_i(y) \leq f_i(\alpha x + [1-\alpha] y) \tag{27}$$

$\forall x, y$ and for every α such that $0 \leq \alpha \leq 1$. In this case, we prove the validity of (26) by induction. If $q_i = 1$, (26) holds trivially. So assume that it holds for $q_i = 1, 2, \ldots, m-1$. Induction assumption implies that:

$$\sum_{j=1}^{m-1} f_i(t_{ij}) \;\leq\; (m-1)\, f_i(\frac{t_{i1} + \ldots + t_{i(m-1)}}{m-1}) \tag{28}$$

Choosing $\alpha = \frac{m-1}{m}$, $x = \frac{t_{i1}+t_{i2}+\ldots+t_{i(m-1)}}{m-1}$, $y = t_{im}$ in (27), we can write:

$$\frac{m-1}{m}\, f_i(\frac{t_{i1} + \ldots + t_{i(m-1)}}{m-1}) + \frac{1}{m}\, f_i(t_{im}) \;\leq\; f_i(\frac{t_{i1} + \ldots + t_{im}}{m}) \tag{29}$$

Combining (28) and (29), we get:

$$\frac{1}{m} \sum_{j=1}^{m} f_i(t_{ij}) \;\leq\; f_i(\frac{t_{i1} + \ldots + t_{im}}{m}) \quad \forall m$$

establishing the validity of (26).