Scheduling Periodic Real-Time Tasks with Heterogeneous Reward Requirements

I-Hong Hou
CSL and Department of CS
University of Illinois
Urbana, IL 61801, USA
ihou2@illinois.edu

P. R. Kumar
Computer Engineering and Systems Group
Department of ECE
Texas A&M University
College Station, TX 77843, USA
prk@tamu.edu

***Abstract*—We study the problem of scheduling periodic real-time tasks which have individual minimum reward requirements. We consider situations where tasks generate jobs that can be provided arbitrary service times before their deadlines, and obtain rewards based on the service times received by the jobs of the task. We show that this model is compatible with the imprecise computation models and the increasing reward with increasing service models. In contrast to previous work on these models, which mainly focus on maximizing the total reward in the system, we additionally aim to fulfill different reward requirements of different tasks. This provides better fairness and also allows fine-grained tradeoff between tasks.**

We first derive a necessary and sufficient condition for a system with reward requirements of tasks to be feasible. We next obtain an off-line feasibility optimal scheduling policy. We then study a sufficient condition for a policy to be feasibility optimal or achieve some approximation bound. This condition serves as a guideline for designing on-line scheduling policy and we obtain a greedy policy based on it. We prove that the on-line policy is feasibility optimal when all tasks have the same periods, and also obtain an approximation bound for the policy under general cases. We test our policies through comparative simulations.

I. Introduction

In classical hard real-time systems, every job needs to be completed before its deadline, or the system suffers from a timing fault. In practice, however, many applications allow approximate results when partially completed jobs only degrade the overall performance rather than causing a fault. Two models that have been proposed to deal with such applications are imprecise computation models [1], [2], and increasing reward with increasing service (IRIS) models [3]. Most work on these models only aims to minimize the total error, or, equivalently, maximize the total reward of the system, without any considerations of fairness [1], [3]–[6]. However, in many applications, rewards of different tasks are not additive, and satisfying individual reward requirements is more important than maximizing total rewards. For example, consider a server that provides video streams to subscribers. Deadline misses will only cause losses on some frames and degrade the video quality, which is usually tolerable as long as such losses happen infrequently. In such an application, a policy that aims to maximize total reward may end up providing perfect video quality for some subscribers while only offering poor quality for others. In contrast, a desirable policy should aim at providing reasonably good quality to all of its subscribers.

In this paper, we describe a model that incorporates both the hard delay bounds for tasks as well as rewards for partially completed jobs, in a system with a set of periodic tasks. The relationship between service times and rewards can be any arbitrary increasing and concave function, and may differ from task to task. We allow each task to have its own individual requirement on the average reward it obtains. We show that both the imprecise computation model and the IRIS model are special cases of our model. Therefore, our model is suitable for handling applications that fit the imprecise computation model or the IRIS model, such as image and speech processing, multimedia applications, real-time monitoring, real-time database, and autonomous control systems [3], [5], [7]. In particular, we show that our model can be used for a server serving multiple video streams. As discussed in the previous paragraph, such an application focuses more on satisfying the video quality requirement for each stream than on maximizing the total rewards in the system.

We first analyze the conditions for feasibility, that is, whether there exists a scheduling policy that meets the individual reward requirements of all tasks in the system. We determine a necessary and sufficient condition for feasibility. We also propose a linear time algorithm for evaluating whether a system is feasible. We further derive an off-line scheduling policy that is feasibility optimal, meaning that if a system that can be strictly fulfilled by some policy then the system is fulfilled by this policy, i.e., this policy fulfills all strictly feasible systems.

We then study the problem of designing on-line scheduling policies. We derive a sufficient condition for a policy to be feasibility optimal, or, serve as an approximation policy with some approximation bound. Using this condition as a guideline, we propose an on-line scheduling policy. We prove that this on-line policy fulfills every feasible system in which the periods are the same for all tasks. We also obtain an approximation bound for this policy when periods of tasks may be different.

This material is based upon work partially supported by USARO under Contract Nos. W911NF-08-1-0238 and W-911-NF-0710287, AFOSR under Contract FA9550-09-0121, and NSF under Contract Nos. CCF-0939370, CNS-1035378, CNS-1035340, and CNS-07-21992.

In addition to theoretical studies, we also conduct simulations to verify our results. We compare our policy against one proposed by Aydin et al [5], which is proved to be an optimal off-line policy that maximizes the total reward in any system. Simulation results suggest that although the policy proposed by [5] achieves maximum total reward, it can result in severe unfairness and does not allow fine-grained tradeoffs between the performances of different tasks.

The rest of the paper is organized as follows. Section II summarizes some existing work and, in particular, introduces the basic concepts in the imprecise computation model and the IRIS model. Section III formally describes our proposed model and discusses how it can capture the imprecise computation model and the IRIS model. Section IV analyzes the necessary and sufficient condition for a system to be feasible, and proposes a linear time algorithm for evaluating feasibility. Section V studies the problem of scheduling jobs and obtains a sufficient condition for a policy to achieve an approximation bound or to be feasibility optimal. Based on this condition, Section VI proposes a simple on-line scheduling policy and analyzes its performance under different cases. Section VII demonstrates our simulation results. Finally, Section VIII concludes this paper.

II. Related Work

The imprecise computation models [1], [2] have been proposed to handle applications in which partially completed jobs are useful. In this model, all jobs consist of two parts: a mandatory part and an optional part. The mandatory part needs to be completed before its deadline, or else the system suffers from a timing fault. On the other hand, the optional part is used to further enhance performance by either reducing errors or increasing rewards. The relations between the errors, or rewards, and the time spent on the optional parts, are described through error functions or reward functions. Chung, Liu, and Lin [1] have proposed scheduling policies that aim to minimize the total average error in the system for this model. Their result is optimal only when the error functions are linear and tasks generate jobs with the same period. Shih and Liu [8] have proposed policies that minimize the maximum error among all tasks in the system when error functions are linear. Feiler and Walker [9] have used feedback to opportunistically schedule optional parts when the lengths of mandatory parts may be time-varying. Mejia-Alvarez, Melhem, and Mosse [4] have studied the problem of maximizing total rewards in the system when job generations are dynamic. Chen et al [10] have proposed scheduling policies that defer optional parts so as to provide more timely response for mandatory parts. Zu and Chang [11] have studied the scheduling problem when optional parts are hierarchical. Aydin et al [5] have proposed an off-line scheduling policy that maximizes total rewards when the reward functions are increasing and concave. Most of these works only concern themselves with the maximization of the total reward in a system. Amirijoo, Hansson, and Son [7] have considered the tradeoff between data errors and transaction errors in a real-time database. The IRIS models can be thought of as special cases of the imprecise computation models where the lengths of mandatory parts are zero. Scheduling policies aimed at maximizing total rewards have been studied for such models [3], [6].

III. System Model

In this section, we introduce our system model. The model is compatible with the imprecise computation model and the IRIS model. Our model is hence suitable for all applications that fit these two models. In particular, we also demonstrate that our model can be used to model a video streaming server.

Consider a system with a set $S = \{A, B, \dots\}$ of real-time tasks. Time is slotted and expressed by $t \in \{1, 2, 3, \dots\}$. Each task X generates a job periodically with period τ_X. A job can be executed multiple times in the period where it is generated; the execution of a job does not mean its completion. The job is removed from the system when the next period begins. In other words, the relative deadline of a job generated by task X is also τ_X. We assume that all tasks in S generate a job at the beginning of the first time slto. We denote a frame as the time between two consecutive time slots where all tasks generate a job. The length of a frame, which we denote by T, is the least common multiple of $\{\tau_X | X \in S\}$. Thus, a frame consists of T/τ_X periods of task X.

As noted above, each job can be executed in an arbitrary number of time slots before its deadline. Each task obtains a certain *reward* each time that its job is executed. The total amount of reward obtained by a task in a period depends on the number of times that its job has been executed in the period. More formally, task X obtains reward $r_X^i \geq 0$ when it executes its job for the i^{th} time in a period. For example, if a job of task X is executed a total of n time slots, then the total reward obtained by task X in this period is $r_X^1 + r_X^2 + \cdots + r_X^n$. We further assume that the marginal reward of executing a job non-strictly decreases as the number of executions increases, that is, $r_X^{i+1} \leq r_X^i$, for all i and X. Thus, the total reward that a task obtains in a period is an increasing and concave function of the number of time slots that its job is executed.

A *scheduling policy* η for the system is one that chooses an *action*, possibly at random, in each time slot, based on past history and system information. The action taken by η at time t is described by $\eta(t) = (X, i)$, meaning that the policy executes the job of task X at time t and that this is the i^{th} time that the job is being executed in the period. Fig. 1 shows an example with two tasks over one frame, which consists of two periods of task A and three periods of task B. In this example, action $(A, 1)$ is executed twice and $(A, 2)$ is executed once in a frame. Thus, the reward obtained by task A in this frame is $2r_A^1 + r_A^2$. On the other hand, the reward obtained by task B in this frame is $3r_B^1$.

The performance of the system is described by the long-term *average reward* per frame of each task in the system.

Definition 1: Let $\tilde{q}_X(k)$ be the total reward obtained by task X in the frame $((k-1)T, kT]$ under some scheduling

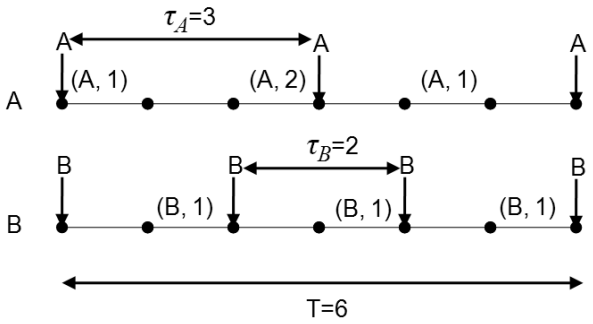


Fig. 1: An example of the system and scheduling policy over a frame of six slots, which consists of two periods of task A and three periods of task B. Arrows in the figure indicate the beginning of a new period.

policy η. The *average reward* of task X is defined as $q_X := \liminf_{k\to\infty} \frac{\sum_{i=1}^{k} \tilde{q}_X(i)}{k}$.

We assume that there is a minimum average reward requirement for each task X, $q_X^* > 0$. We wish to verify whether a system is *feasible*, that is, whether each task can have its minimum average reward requirement satisfied.

Definition 2: A system is *fulfilled* by a scheduling policy η if, under η, $q_X \geq q_X^*$ with probability 1, for all $X \in S$.

Definition 3: A system is *feasible* if there exists some scheduling policy that fulfills it.

A natural metric to evaluate a scheduling policy is the set of systems that the policy fulfills. For ease of discussion, we only consider systems that are *strictly feasible*.

Definition 4: A system with minimum reward requirements $[q_X^* | X \in S]$ is *strictly feasible* if there exists some $\epsilon > 0$ such that the same system with minimum reward requirements $[(1+\epsilon)q_X^*]$ is feasible.

Definition 5: A scheduling policy is *feasibility optimal* if it fulfills all strictly feasible systems.

We limit discussions on strictly feasible systems only to simplify the proof in Theorem 6. Since ϵ can be chosen to be any arbitrarily small positive number, this limitation does not have practical importance.

Moreover, since the overhead for computing a feasibility optimal policy may be too high in certain scenarios, we also need to consider simple approximation policies.

Definition 6: A scheduling policy is a *p-approximation policy*, $p \geq 1$, if it fulfills all systems with minimum reward requirements $[q_X^*]$ such that the same system with minimum reward requirements $[pq_X^*]$ is strictly feasible.

A. Extensions for Imprecise Computation Models

In this section, we discuss how our proposed model can be used to handle imprecise computation models and IRIS models. In such models, a task consists of two parts: a mandatory part and an optional part. The mandatory part is required to be completed in each period, or else the system fails. After the mandatory part is completed, the optional part can be executed to improve performance. The more optional parts executed for a task, the more rewards it gets.

Let m_X be the length of the mandatory part of task X, that is, it is required that each job of X is executed at least m_X time slots in each of its period. Let o_X be the length of the optional part of task X. To accommodate this scenario, we define a symbolic value M with the following arithmetic reminiscent of the "Big-M Method" in linear programming: $0\times M = 0$, $aM+bM = (a+b)M$, $a\times(bM) = (ab)M$, $aM + c > bM + d$, if $a > b$, and $aM + c > aM + d$ if $c > d$, for all real numbers a, b, c, d. Loosely speaking, M can be thought of as a huge positive number. For each task X, we set $r_X^1 = r_X^2 = \cdots = r_X^{m_X} = M$, we then set $r_X^{m_X+1}, r_X^{m_X+2}, \ldots, r_X^{m_X+o_X}$ according to the rewards obtained by X for its optional part, and $r_X^i = 0$, for all $i > m_X+o_X$. The minimum reward requirement of task X is set to be $\frac{T}{\tau_X} m_X M + \hat{q}_X^*$ with $\hat{q}_X^* \geq 0$. Thus, a scheduling policy that fulfills such a system is guaranteed to complete each mandatory part with probability one.

We now discuss how to apply our model to a video streaming server. In typical MPEG videos, there are three types of frames: I-frames, P-frames, and B-frames. A video is composed of groups of frames (GOFs) where a GOF consists of a frame sequence of I,B,B,P,B,B,P,B,B,P,.... We can treat a video stream as a task in the system and its GOFs as the jobs of the task. The I-frames consist of the complete information of the frame and can be decoded independently. On the other hand, P-frames and B-frames may require the information of the I-frame and P-frames in the GOF to be decoded. If an I-frame is lost, none of the succeeding frames in the GOF can be decoded. Thus, the I-frames should be treated as mandatory parts. On the other hand, a lost B-frame does not cause problems for other frames and only results in minor video quality degradation. Thus, B-frames can be treated as optional parts and the server can drop B-frames when it is overloaded. The reward for a B-frame reflects the additional video quality it provides. Finally, while a lost P-frame can also cause problems to other P-frames and B-frames, its influence is not as severe as a lost I-frame. P-frames can therefore be treated as either mandatory parts or optional parts, depending on the requirements of subscribers of the videos. Details of MPEG encoding can be found in [12].

IV. Feasibility Analysis

In this section, we establish a necessary and sufficient condition for a system to be feasible. Consider a feasible system that is fulfilled by a policy η. Suppose that, on average, there are f_X^i periods of task X in a frame in which the action (X, i) is taken by η. The average reward of task X can then be expressed as $q_X = \sum_{i=1}^{\tau_X} f_X^i r_X^i$. We can immediately obtain a necessary condition for feasibility.

Lemma 1: A system with a set of tasks $S = \{A, B, \ldots\}$ is feasible only if there exists $\{f_X^i | X \in S, 1 \leq i \leq \tau_X\}$ such that

$$q_X^* \leq \sum_{i=1}^{\tau_X} f_X^i r_X^i, \qquad \forall X \in S, \tag{1}$$

$$0 \leq f_X^i \leq T/\tau_X, \qquad \forall X \in S, 1 \leq i \leq \tau_X, \tag{2}$$

$$\sum_{X\in S} \sum_{i=1}^{\tau_X} f_X^i \leq T. \tag{3}$$

Proof: Condition (1) holds because task X requires that $q^*_X \leq q_X$. Condition (2) holds because there are T/τ_X periods of task X in a frame and thus f^i_X is upper-bounded by T/τ_X. Finally, the total average number of time slots that the system executes one of the jobs in a frame can be expressed as $\sum_{X\in S}\sum_{i=1}^{\tau_X} f^i_X$, which is upper-bounded by the number of time slots in a frame, T. Thus, condition (3) follows. ■

Next, we show that the conditions (1)–(3) are also sufficient for feasibility. In practice, we also have $f^i_X \geq f^{i+1}_X$, since we need to schedule the action (X,i) before the action $(X,i+1)$ can be taken. However, we note that we do not need this constraint when evaluating feasibility. To prove that the conditions (1)–(3) are sufficient, we first show that the polytope, which contains all points $f = (f^1_A, f^2_A, \ldots, f^{\tau_A}_A, f^1_B, \ldots)$ that satisfy conditions (2) and (3), is a convex hull of several integer points. We then show that for all integer points $n = (n^1_A, n^2_A, \ldots, n^{\tau_A}_A, n^1_B, \ldots)$ in the polytope, there is a schedule under which the reward obtained by task X is at least $\sum_{i=1}^{\tau_X} n^i_X r^i_X$. We then prove sufficiency using these two results.

Define a matrix $H = [h_{i,j}]$ with $2\sum_{X\in S}\tau_X + 1$ rows and $\sum_{X\in S}\tau_X$ columns as follows:

$$h_{i,j} = \begin{cases} 1, & \text{if } i = 1, \\ 1, & \text{if } i = 2j, \\ -1, & \text{if } i = 2j+1, \\ 0, & \text{else.} \end{cases} \tag{4}$$

Define $b = [b_i]$ to be a column vector with $2\sum_{X\in S}\tau_X + 1$ elements so that $b_1 = T$; the first τ_A elements with even indices are set to T/τ_A, that is, $b_2 = b_4 = \cdots = b_{2\tau_A} = T/\tau_A$; the next τ_B elements with even indices are set to T/τ_B, and so on. All other elements are set to 0. For example, the system shown in Fig. 1 would have

$$H = \begin{bmatrix} 1, & 1, & 1, & 1, & \ldots \\ 1, & 0, & 0, & 0, & \ldots \\ -1, & 0, & 0, & 0, & \ldots \\ 0, & 1, & 0, & 0, & \ldots \\ 0, & -1, & 0, & 0, & \ldots \\ 0, & 0, & 1, & 0, & \ldots \\ 0, & 0, & -1, & 0, & \ldots \\ & & \vdots & & \end{bmatrix}, b = \begin{bmatrix} T \\ T/\tau_A \\ 0 \\ T/\tau_A \\ 0 \\ \vdots \\ T/\tau_B \\ 0 \\ \vdots \end{bmatrix}.$$

Thus, conditions (2) and (3) can be described as $Hf \leq b$. Theorem 5.20 and Theorem 5.24 in [13] show the following:

Theorem 1: The polytope defined by $\{f|Hf \leq b\}$, where b is an integer vector, is a convex hull of several integer points if for every subset R of rows in H, there exists a partition of $R = R_1 \cup R_2$ such that for every column j in H, we have $\sum_{i_1\in R_1} h_{i_1,j} - \sum_{i_2\in R_2} h_{i_2,j} \in \{1, 0, -1\}$.[1]

Since all elements in b are integers, we obtain the following:

[1]Such a matrix H is called a *totally unimodular matrix* in combinatorial optimization theory.

Theorem 2: The polytope defined by $\{f|Hf \leq b\}$, where H and b are derived using conditions (2) and (3) as above, is a convex hull of several integer points.

Proof: Let $R = \{r_1, r_2, \ldots\}$ be the indices of some subset of rows in H. If the first row is in R, we choose $R_1 = \{r | r \in R, r \text{ is odd}\}$ and $R_2 = \{r | r \in R, r \text{ is even}\}$. Since for all columns j in H, $h_{1,j} = 1$, $h_{2j,j} = 1$, $h_{2j+1,j} = -1$, and all other elements in column j are zero, we have $\sum_{i_1\in R_1} h_{i_1,j} - \sum_{i_2\in R_2} h_{i_2,j} \in \{1, 0, -1\}$. On the other hand, if the first row is not in R, we choose $R_1 = R$ and $R_2 = \varnothing$. Again, we have $\sum_{i_1\in R_1} h_{i_1,j} - \sum_{i_2\in R_2} h_{i_2,j} = \sum_{i_1\in R_1} h_{i_1,j} \in \{1, 0, -1\}$. Thus, by Theorem 1, the polytope defined by $\{f|Hf \leq b\}$ is a convex hull of several integer points. ■

Next we show that all integer points in the polytope can be carried out by some scheduling policy as follows. As noted before, in the following theorem, we do not require that $n^i_X \geq n^{i+1}_X$.

Theorem 3: Let $n = (n^1_A, n^2_A, \ldots, n^{\tau_A}_A, n^1_B, \ldots)$ be an integer point in the polytope $\{f|Hf \leq b\}$. Then, there exists a scheduling policy so that $q_X \geq \sum_{i=1}^{\tau_X} n^i_X r^i_X$.

Proof: We prove this theorem by finding a schedule with $q_X \geq \sum_{i=1}^{\tau_X} n^i_X r^i_X$. Ideally, we want to schedule the action (X,i) n^i_X times in a frame. We can schedule at most one (X,i) action in each period of X. Further, we can schedule at most one action in each time slot. We construct a flow network based on these constraints. In the flow network, there are three layers of nodes between the source r and sink s, which we refer to as $L1, L2$, and $L3$, respectively. In $L1$, each node represents an action (X,i). There is an edge from the source to the node representing (X,i) with capacity n^i_X, since we want to schedule (X,i) n^i_X times. In $L2$, we have T/τ_X nodes for each task X, where each of these nodes represents a period of X and these nodes are named as $[X,1], [X,2], \ldots, [X, T/\tau_X]$. There is an edge from the node in $L1$ representing (X,i) to the node $[X,j]$ with capacity 1, for all X, i, and j. This represents the restriction that there is at most one (X,i) action in each period of X. In $L3$, there is one node for each time slot in the frame. Nodes in $L3$ are named as $\{1\}, \{2\}, \ldots, \{T\}$. There is an edge with infinite capacity from the node $[X,j]$ in $L2$ to $\{t\}$ in $L3$, for all $t \in ((j-1)\tau_X, j\tau_X]$. This shows that the j^{th} period of X consists of the time slots $((j-1)\tau_X, j\tau_X]$. Finally, there is an edge from each node in $L3$ to the sink s with capacity 1, showing that there is at most one action in each time slot. Fig. 2a shows the flow network derived from a system with two tasks, A and B, with $\tau_A = 3$, $\tau_B = 2$, $n^1_A = 1$, $n^2_A = 2$, $n^3_A = 0$, $n^1_B = 2$, and $n^2_B = 1$.

We now show that the maximum flow of this network is $\sum_i \sum_X n^i_X$. By the max-flow min-cut theorem, it suffices to show that every cut of the network has capacity at least $\sum_i \sum_X n^i_X$. For any given cut, let C_r be the complement that contains the source r and C_s be the complement that contains the sink s. The capacity of the cut is the sum of capacities of edges from C_r to C_s. We consider several different types of cuts. First, consider a cut where some nodes in $L3$ belong to C_r and some other nodes in $L3$ belong to C_s. There exists t such that the node $\{t\}$ is

in C_r and the node $\{t+1\}$ is in C_s, where $t+1 \leq T$. There must be a task X which has a period containing both time slots t and $t+1$, or otherwise t is a common multiple of $\{\tau_X | X \in S\}$ that is smaller that T, which yields a contradiction. Thus, there exists a node $[X, j]$ in $L2$ which has edges toward both $\{t\}$ and $\{t+1\}$, one of which must be between C_r and C_s. The capacity of this cut is infinite since the capacities of edges between $L2$ and $L3$ are infinite. Next, consider a cut where all nodes in $L3$ are in the same complement but some nodes in $L2$ are in the other complement. The capacity of this cut is also infinite because every node in $L2$ has at least one edge, whose capacity is infinite, towards a node in $L3$. It remains to consider cases where the nodes in $L2$ and $L3$ are all in the same complement. If they are in C_r, the capacity of the cut is T since there is one edge with capacity one from each node in $L3$ to the sink. The capacity of this cut is then greater or equal to $\sum_i \sum_X n_X^i$ by (3). Finally, if all nodes in $L2$ and $L3$ are in C_s, the capacity of the cut is $\sum_{(X,i)\in C_r} T/\tau_X + \sum_{(X,i)\in C_s} n_X^i \geq \sum_i \sum_X n_X^i$, where the inequality follows from (2), and equality is achieved when all nodes in $L1$ are in C_s. In sum, the maximum flow of this network is $\sum_i \sum_X n_X^i$.

Since the capacities of all edges in the network are integers, there exists an integer flow whose value is $\sum_i \sum_X n_X^i$. We construct a schedule based on this integer flow. We schedule the action (X, i) at time slot t if there is a flow from (X, i) to $\{t\}$. This schedule has the following properties: each action (X, i) is scheduled n_X^i times; each action (X, i) is scheduled at most once in a period of X; there is at most one action in each time slot. Fig. 2b shows an example of the resulting schedule. Note that, under this policy, there are time slots where the policy schedules an action (X, i), but it is instead the j^{th} time that the job of X is executed in the period. Thus, we may need to renumber these actions as in Fig. 2c and define $\bar{n}_X^i$ as the actual number of times that the action (X, i) is executed in the frame. In the example of Fig. 2, we have $\bar{n}_A^1 = 2, \bar{n}_A^2 = 1, \bar{n}_B^1 = 3$, and $\bar{n}_B^2 = 0$. Since the policy does not schedule two identical actions in a period, we have $\sum_{i=1}^{k} \bar{n}_X^i \geq \sum_{i=1}^{k} n_X^i$, for all k and $X \in S$. Thus, $q_X = \sum_{i=1}^{\tau_X} \bar{n}_X^i r_X^i \geq \sum_{i=1}^{\tau_X} n_X^i r_X^i$, since $r_X^1 \geq r_X^2 \geq \ldots$, for all $X \in S$.

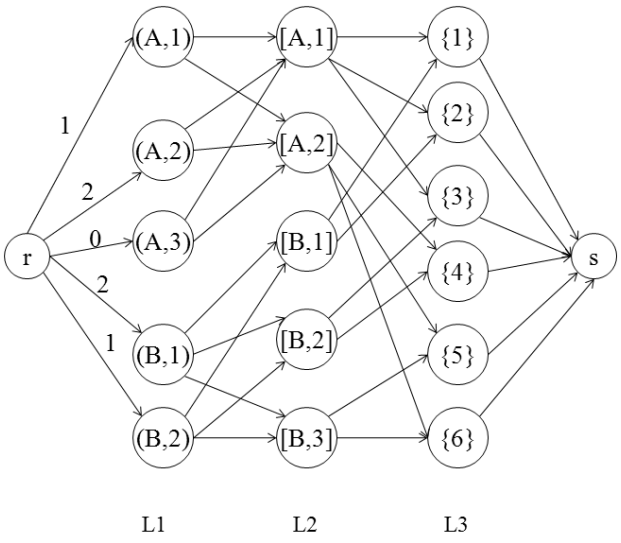
■

Now we can derive the necessary and sufficient condition for a system to be feasible.

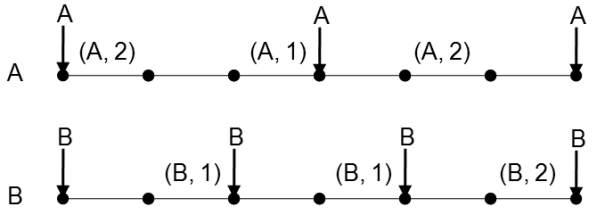
Theorem 4: A system with a set of tasks $S = \{A, B, \ldots\}$ is feasible if and only if there exists $\{f_X^i | X \in S, 1 \leq i \leq \tau_X\}$ such that (1) - (3) are satisfied.

Proof: Lemma 1 has established that these conditions are necessary. It remains to show that they are also sufficient. Suppose there exists $f = \{f_X^i | X \in S, 1 \leq i \leq \tau_X\}$ that satisfies (1) - (3). Then Theorem 2 shows that there exists integer vectors $n[1], n[2], \ldots, n[v]$ such that each of them satisfies (1) - (3), and $f = \sum_{u=1}^{v} \alpha_u n[u]$, where α_u's are positive numbers with $\sum_{u=1}^{v} \alpha_u = 1$. Let η_u be the scheduling policy for the integer vector $n[u]$ as in the proof of Theorem 3, for each u. Theorem 3 has shown that for each u, the average reward obtained by X under η_u, $q_X[\eta_u]$, is at least $\sum_{i=1}^{\tau_X} n[u]_X^i r_X^i$. Finally, we can design a weighted round robin policy that switches among the policies $\eta_1, \eta_2, \ldots, \eta_v$, so that the proportion of frames that policy η_u is chosen is α_u. The average reward obtained by X is hence $q_X = \sum_u \alpha_u q_X[\eta_u] \geq \sum_u \alpha_u (\sum_{i=1}^{\tau_X} n[u]_X^i r_X^i) = \sum_{i=1}^{\tau_X} f_X^i r_X^i \geq q_X^*$. Thus, this policy fulfills the system and so the conditions are also sufficient. ■

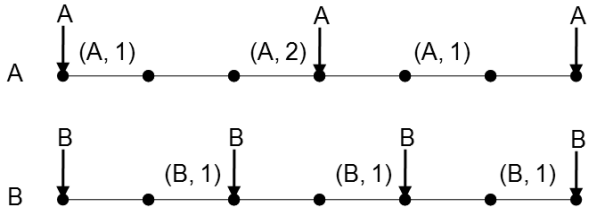
Using Theorem 4, checking whether a system is feasible can be done by any linear programming solver. The computational overhead for checking feasibility can be further reduced by using the fact that $r_X^i \geq r_X^j$, for all $i < j$. Given a system and $\{f_X^i\}$ that satisfy conditions (1) - (3) with $f_Y^j < \frac{T}{\tau_Y}$ and $f_Y^k > 0$ for some $j < k$ and $Y \in S$, let $\delta = \min\{\frac{T}{\tau_Y} - f_Y^j, f_Y^k\}$. Construct $\{\hat{f}_X^i\}$ such



(a) The resulting flow network. Capacities for edges between the source and $L1$ are marked, while those for other edges are not shown for readability.



(b) The actions scheduled before renumbering



(c) The actions scheduled after renumbering

Fig. 2: An example of the scheduling policy in Theorem 3.

that $\hat{f}_Y^j = f_Y^j + \delta$, $\hat{f}_Y^k = f_Y^k - \delta$, and $\hat{f}_X^i = f_X^i$ for all other elements. Then $\{\hat{f}_X^i\}$ also satisfies conditions (1) - (3). Based on this observation, we derive an algorithm for checking feasibility, described in Algorithm 1. This essentially transfers slots from less reward earning actions to more reward earning actions. The running time of this algorithm is $O(\sum_{X\in S}\tau_X)$. Since a specification of a system involves at least the $\sum_{X\in S}\tau_X$ variables of $\{r_X^i\}$, Algorithm 1 is essentially a linear time algorithm.

Algorithm 1 Feasibility Checker

Require: S, $\{\tau_X | X \in S\}$, $\{r_X^i | X \in S, 1 \le i \le \tau_X\}$, $\{q_X^* | X \in S\}$

```
for X ∈ S do
    if q*_X > (T/τ_X) Σ_{i=1}^{τ_X} r^i_X then
        return Infeasible
for X ∈ S do
    i ← 1
    while q*_X > 0 do
        if q*_X > (T/τ_X) r^i_X then
            f^i_X ← T/τ_X
            q*_X ← q*_X − (T/τ_X) r^i_X
        else
            f^i_X ← q*_X / r^i_X
            q*_X ← 0
        i ← i + 1
if Σ_{X∈S} Σ_{τ_X}^{i=1} f^i_X ≤ T then
    return Feasible
else
    return Infeasible
```

In addition to evaluating feasibility, the proof of Theorem 4 also demonstrates an off-line feasibility optimal policy. In many scenarios, however, on-line policies are preferred. In the next section, we introduce a guideline for designing scheduling policies that is suggestive of simple on-line policies.

V. Designing Scheduling Policies

In this section, we study the problem of designing scheduling policies. We establish sufficient conditions for a policy to be either feasibility optimal or p-approximately so.

We start by introducing a metric to evaluate the performance of a policy η. Recall that $\tilde{q}_X(k)$ is the total reward obtained by task X during the frame $((k-1)T, kT]$ and the average reward of X is $q_X = \liminf_{k\to\infty} \frac{\sum_{i=1}^k \tilde{q}_X(i)}{k}$. We now define the *debt* of task X.

Definition 7: The *debt* of task X in the frame $((k-1)T, kT]$, $d_X(k)$, is defined recursively as follows:

$$d_X(0) = 0,$$
$$d_X(k) = [d_X(k-1) + q_X^* - \tilde{q}_X(k)]^+, \forall k > 0,$$

where $x^+ := \max\{0, x\}$.

Lemma 2: A system is fulfilled by a policy η if $\lim_{k\to\infty} d_X(k)/k = 0$ with probability 1.

Proof: We have $d_X(k) \ge kq_X^* - \sum_{i=1}^k \tilde{q}_X(i)$ and $d_X(k)/k \ge q_X^* - \frac{1}{k}\sum_{i=1}^k \tilde{q}_X(i)$. Thus, if $\lim_{k\to\infty} d_X(k)/k = 0$, then $q_X = \liminf_{k\to\infty} \frac{\sum_{i=1}^k \tilde{q}_X(i)}{k} \ge q_X^*$ and the system is fulfilled. ■

We can describe the *state* of the system in the k^{th} frame by the debts of tasks, $[d_X(k)|X \in S]$. Consider a policy that schedules jobs solely based on the requirements and the state of the system. The evolution of the state of the system can then be described by a Markov chain. In each frame k, the Markov chain visits the state $[d_X(k)|X \in S]$. The Markov chain is said to be *positive recurrent* if it visits each reachable state infinitely many times as $k \to \infty$, and the mean time between two consecutive visits to any particular reachable state is finite.

Lemma 3: Suppose the evolution of the state of a system can be described by a Markov chain under some policy η. The system is fulfilled by η if this Markov chain is positive recurrent.

Proof: Since the Markov chain is positive recurrent, the state $\{d_X(k) = 0, \forall X \in S\}$ is visited infinitely many times. Further, assuming that the system is in this state at frames $k_1, k_2, k_3, \ldots$, then $[k_{n+1} - k_n]$ is a series of i.i.d. random variables with finite mean. Let I_n be the indicator variable that there exists some $\tilde{k}_n$ between frame k_n and frame k_{n+1} such that $d_X(\tilde{k}_n)/\tilde{k}_n > \delta$ for some $X \in S$, for some arbitrary $\delta > 0$. Let $q_{max}^* = \max_{X\in S} q_X^*$. If $I_n = 1$, we have that $\tilde{k}_n \ge k_n \ge n$ and thus $d_X(\tilde{k}_n) > n\delta$, for some $X \in S$. Since $d_X(k)$ can be incremented by at most q_{max}^* in a frame and $d_X(k_n) = 0$, we have $\tilde{k}_n - k_n > n\delta/q_{max}^*$ and $k_{n+1} - k_n > \tilde{k}_n - k_n > n\delta/q_{max}^*$. Thus,

$$\begin{aligned} Prob\{I_n = 1\} &< Prob\{k_{n+1} - k_n > n\delta/q_{max}^*\} \\ &= Prob\{k_2 - k_1 > n\delta/q_{max}^*\}, \end{aligned}$$

and

$$\begin{aligned} \sum_{n=1}^\infty Prob\{I_n = 1\} &< \sum_{n=1}^\infty Prob\{k_2 - k_1 > n\delta/q_{max}^*\} \\ &\le E[k_2 - k_1] < \infty, \end{aligned}$$

By the Borel-Cantelli Lemma, the probability that $I_n = 1$ for infinitely many n's is zero, and so is the probability that $d_X(k)/k > \delta$ for infinitely many k's. Thus, $\limsup_{k\to\infty} d_X(k)/k < \delta$ with probability 1, for all $X \in S$ and any arbitrary $\delta > 0$. Finally, we have $\lim_{k\to\infty} d_X(k)/k = 0$ with probability 1 since $d_X(k) \ge 0$ by definition. ■

Based on the above lemmas, we determine a sufficient condition for a policy to be a p-approximation policy. The proof is based on the Foster-Lyapunov Theorem:

Theorem 5 (Foster-Lyapunov Theorem): Consider a Markov chain with state space $\mathcal{D}$. Let $D(k)$ be the state of the Markov chain at the k^{th} step. If there exists a non-negative function $L : \mathcal{D} \to R$, a positive number δ, and a finite subset $\mathcal{D}_0$ of $\mathcal{D}$ such that:

$$\begin{aligned} E[L(D(k+1)) - L(D(k))|D(k)] &\le -\delta, \text{ if } D(k) \notin \mathcal{D}_0, \\ E[L(D(k+1))|D(k)] &< \infty, \text{ if } D(k) \in \mathcal{D}_0, \end{aligned}$$

then the Markov chain is positive recurrent. ■

Theorem 6: A policy η is a p-approximation policy, for some $p > 1$, if it schedules jobs solely based on the

requirements and the state of a system, and, for each k, the following holds:

$$\sum_{X\in S}\tilde{q}_X(k)d_X(k) \geq (\max_{[q_X]:\ [q_X] \text{ is feasible}} \sum_{X\in S} q_X d_X(k))/p.$$

Proof: Consider a system with minimum reward requirements $[q_X^*]$ such that the same system with minimum reward requirements $[pq_X^*]$ is also strictly feasible. By Lemma 3, it suffices to show that under the policy η, the resulting Markov chain is positive recurrent. Consider the Lyapunov function $L(k) := \sum_{X\in S} d_X^2(k)/2$. The Lyapunov drift function can be written as:

$$\Delta L(k+1) := L(k+1) - L(k) = \frac{1}{2}\sum_{X\in S}[d_X^2(k+1) - d_X^2(k)]$$
$$\leq \sum_{X\in S}(q_X^* - \tilde{q}_X(k))d_X(k) + C,$$

where C is a bounded constant. Since $[pq_X^*]$ is also strictly feasible and $d_X(k) \geq 0$, there exists $\epsilon > 0$ such that $(1+\epsilon)\sum_{X\in S} pq_X^* d_X(k) \leq \max_{[q_X]:\ [q_X] \text{ is feasible}} \sum_{X\in S} q_X d_X(k)$, and hence $(1+\epsilon)\sum_{X\in S} q_X^* d_X(k) \leq \sum_{X\in S}\tilde{q}_X(k)d_X(k)$. Thus, we have

$$\Delta L(k+1) \leq -\epsilon\sum_{X\in S} q_X^* d_X(k) + C. \quad (5)$$

Let $\mathcal{D}_0$ be the set of states $[d_X(k)|X \in S]$ with $\sum_{X\in S} q_X^* d_X(k) < (C+\delta)/\epsilon$, for some positive finite number δ. Then, $\mathcal{D}_0$ is a finite set (since $q_X^* > 0$ for all $x \in S$), with $\Delta L(k+1) < -\delta$ when the state of frame k is not in $\mathcal{D}_0$. Further, since $d_X(k)$ can be increased by at most q_X^* in each frame, $L(k+1)$ is finite if the state of frame k is in $\mathcal{D}_0$. By Theorem 5, this Markov chain is positive recurrent and policy η fulfills this system. ∎

Since a 1-approximation policy is also a feasibility optimal one, a similar proof yields the following:

Theorem 7: A policy η fulfills a strictly feasible system if it maximizes $\sum_{X\in S}\tilde{q}_X(k)d_X(k)$ among all feasible $[q_X]$ in every frame k. It is a feasibility optimal policy if the above holds for all strictly feasible systems.

VI. An On-Line Scheduling Policy

While Section V has described a sufficient condition for designing feasibility optimal policies, the overhead for computing such a feasibility optimal scheduling policy may be too high to implement. In this section, we introduce a simple on-line policy. We also analyze the performance of this policy under different scenarios.

Theorem 7 has shown that a policy that maximizes $\sum_{X\in S}\tilde{q}_X(k)d_X(k)$ among all feasible $[q_X]$ in every frame k is feasibility optimal. The on-line policy follows this guideline greedily. Assume that, at some time t in frame k, task X has already been scheduled j_X times in its period. The on-line policy then schedules the task Y so that $r_Y^{j_Y+1}d_Y(k)$ is maximized among all $X \in S$. A more detailed description of this policy, which we call the *Greedy Maximizer*, is shown in Algorithm 2.

Next, we evaluate the performance of the Greedy Maximizer. We show that this policy is feasibility optimal if the periods of all tasks are the same, and that it is a 2-approximation policy in general.

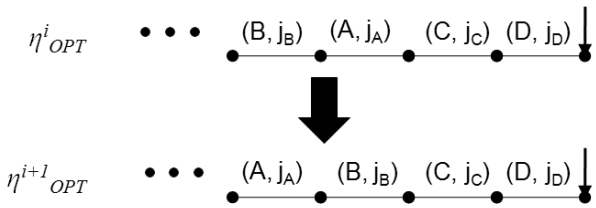
Algorithm 2 Greedy Maximizer

Require: S, $\{\tau_X|X \in S\}$, $\{r_X^i|X \in S, 1 \leq i \leq \tau_X\}$, $\{q_X^*|X \in S\}$
1: $T \leftarrow$ least common multiplier of $\{\tau_X|X \in S\}$
2: **for** $X \in S$ **do**
3: $\quad d_X \leftarrow 0$
4: $t \leftarrow 0$
5: **loop**
6: $\quad t \leftarrow t+1$
7: $\quad$ **if** $t \mod T = 1$ {A new frame} **then**
8: $\quad\quad$ **for** $X \in S$ **do**
9: $\quad\quad\quad d_X \leftarrow [d_X + q_X^* - \tilde{q}_X]^+$
10: $\quad\quad\quad \tilde{q}_X \leftarrow 0$
11: $\quad\quad\quad j_X \leftarrow 0$
12: $\quad$ **for** $X \in S$ **do**
13: $\quad\quad$ **if** $t \mod \tau_X = 1$ {A new period for X} **then**
14: $\quad\quad\quad j_X \leftarrow 0$
15: $\quad Y \leftarrow \arg\max_{X\in S} r_X^{j_X+1} d_X$
16: $\quad j_Y \leftarrow j_Y + 1$
17: $\quad \tilde{q}_Y \leftarrow \tilde{q}_Y + r_Y^{i_Y}$
18: $\quad$ execute the job of Y at time t

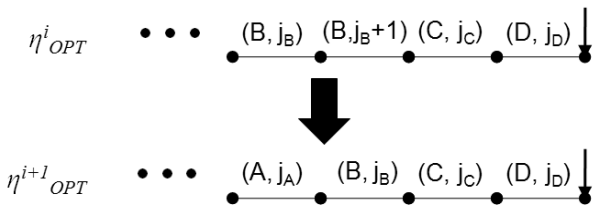
Theorem 8: The Greedy Maximizer fulfills all strictly feasible systems with $\tau_X \equiv \tau$, for all $X \in S$.

Proof: It suffices to prove that the Greedy Maximizer indeed maximizes $\sum_{X\in S} d_X(k)\tilde{q}_X(k)$ in every frame. Suppose at some frame k, the debts are $\{d_X(k)\}$ and the schedule generated by the Greedy Maximizer is $\eta_{GM}(t)$, $t \in (kT, (k+1)T]$. Let $GM := \sum_{X\in S} d_X(k)\tilde{q}_X(k)$ when η_{GM} is applied. Consider another schedule, $\eta_{OPT}(t)$, that achieves $\text{Max}\sum_{X\in S} d_X(k)\tilde{q}_X(k) =: OPT$ in this frame. We need to show that $GM \geq OPT$.

We will modify $\eta_{OPT}(t)$ slot by slot until it is the same as η_{GM}. Let $\eta_{OPT}^i(t)$ be the schedule after we have made sure $\eta_{OPT}^i(t) = \eta_{GM}(t)$ for all t between kT and i, and let $OPT^i := \sum_{X\in S} d_X(k)\tilde{q}_X(k)$ when $\eta_{OPT}^i(t)$ is applied. We then have $\eta_{OPT} \equiv \eta_{OPT}^{kT}$ and $\eta_{GM} \equiv \eta_{OPT}^{(k+1)T}$. The process of modification is as follows: If $\eta_{OPT}^i(i+1) = \eta_{GM}(i+1)$, then we do not need to modify anything and we simply set $\eta_{OPT}^{i+1} \equiv \eta_{OPT}^i$. On the other hand, if $\eta_{OPT}^i(i+1) \neq \eta_{GM}(i+1)$, say, $\eta_{GM}(i+1) = (A, j_A)$ and $\eta_{OPT}^i(i+1) = (B, j_B)$, then we modify $\eta_{OPT}^i(t)$ under two different cases. The first case is that η_{OPT}^i is going to schedule the action (A, j_A) some time after $i+1$ in this frame. In this case η_{OPT}^{i+1} is obtained by switching the two actions (A, j_A) and (B, j_B) in η_{OPT}^i. One such example is shown in Fig. 3a. Since interchanging the order of actions does not alter the value of $\sum_{X\in S} d_X(k)\tilde{q}_X(k)$, we have $OPT^{i+1} = OPT^i$ for this case. The second case is that η_{OPT}^i does not schedule the action (A, j_A) in the frame. Then η_{OPT}^{i+1} is obtained by setting $\eta_{OPT}^{i+1}(i+1) = (A, j_A)$ and scheduling the same jobs as η_{OPT}^i for all succeeding time slots. Since the Greedy Maximizer schedules (A, j_A) in this slot, we have $r_A^{j_A} d_A(k) \geq r_B^{j_B} d_B(k)$. Also, for all succeeding time slots, if job B is scheduled, then the reward for that slot is going to be increased since the number of executions of job B has been decreased by 1;

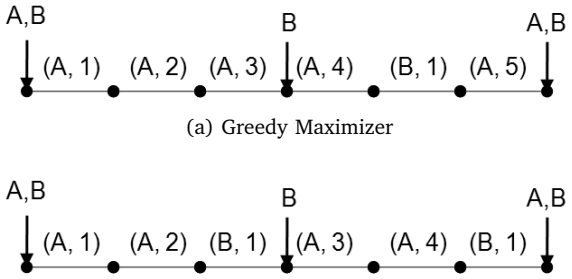


(a) First case



(b) Second case

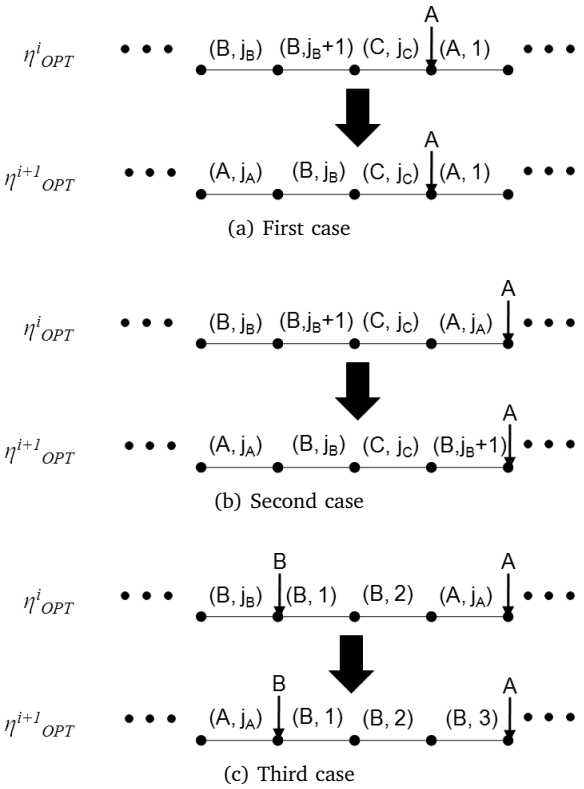
Fig. 3: Examples of modification in Theorem 8



(a) Greedy Maximizer

(b) Feasibility optimal scheduler

Fig. 4: An example of the resulting schedule from the Greedy Maximizer, and a feasibility optimal scheduler, respectively, in Example 1.



(a) First case

(b) Second case

(c) Third case

Fig. 5: Examples of modification in Theorem 9

if a job C other than A and B is scheduled, then the reward for that slot is not altered by the modification. Fig. 3b illustrates one such example. In sum, we have that $OPT^{i+1} \geq OPT^i$.

We have established that $OPT^{i+1} \geq OPT^i$ for all $i \in [kT, (k+1)T]$. Since $OPT = OPT^{kT}$ and $GM = OPT^{(k+1)T}$, we have $GM \geq OPT$ and thus the Greedy Maximizer indeed maximizes $\sum_{X \in S} d_X(k)\tilde{q}_X(k)$. ■

However, when the periods of tasks are not the same, the Greedy Maximizer does not always maximize $\sum_{X \in S} d_X(k)\tilde{q}_X(k)$ and thus may not be feasibility optimal. An example is given below.

Example 1: Consider a system with two tasks, A and B, with $\tau_A = 6$, $\tau_B = 3$. Assume that $r_A^1 = r_A^2 = r_A^3 = r_A^4 = 100$, $r_A^5 = r_A^6 = 1$, $r_B^1 = 10$, and $r_B^2 = r_B^3 = 0$. Suppose, at some frame k, $d_A(k) = d_B(k) = 1$. The Greedy Maximizer would schedule jobs as in Fig. 4a, and yield $d_A(k)\tilde{q}_A(k) + d_B(k)\tilde{q}_B(k) = 411$. On the other hand, a feasibility optimal scheduler would schedule jobs as in Fig. 4b, and yield $d_A(k)\tilde{q}_A(k) + d_B(k)\tilde{q}_B(k) = 420$. ■

Although the Greedy Maximizer is not feasibility optimal, we can still derive an approximation bound for this policy.

Theorem 9: The Greedy Maximizer is a 2-approximation policy.

Proof: The proof is similar to that of Theorem 8. Define $\eta_{GM}, \eta_{OPT}, \eta^i_{OPT}, GM, OPT$, and OPT^i in the same way as in the proof of Theorem 8. By Theorem 6, it suffices to show that $GM \geq OPT/2$.

We obtain η^i_{OPT} as follows: If $\eta_{GM}(i+1) = \eta^i_{OPT}(i+1)$, then we set $\eta^{i+1}_{OPT} \equiv \eta^i_{OPT}$. If $\eta_{GM}(i+1) = (A, j_A) \neq (B, j_B) = \eta^i_{OPT}(i)$, then we consider three possible cases. The first case is that the job (A, j_A) is not scheduled by η^i_{OPT} in this period of A. In this case, we set $\eta^{i+1}_{OPT}(i+1) = (A, j_A)$ and use the same schedule as η^i_{OPT} for all succeeding time slots. An example is shown in Fig. 5a. The same analysis in Theorem 8 shows that $OPT^{i+1} \geq OPT^i$. The second case is that the job (A, j_A) is scheduled by η^i_{OPT} in this period of A and there is no deadline of B before the deadline of A. In this case, we obtain η^{i+1}_{OPT} by switching the jobs (A, j_A) and (B, j_B) in η^i_{OPT}. An example is shown in Fig. 5b. We have $OPT^i = OPT^{i+1}$ for this case. The last case is that the job (A, j_A) is scheduled by η^i_{OPT} in this period of A, and there is a deadline of B before the deadline of A. In this case also, we obtain η^{i+1}_{OPT} by switching the two jobs and renumbering these jobs if necessary. The rewards obtained by all tasks other than B are not altered by this modification. However, as in the example shown in Fig. 5c, the job (B, j_B) in η^i_{OPT} may become a job (B, j'_B) in η^{i+1}_{OPT} with $j'_B > j_B$. Thus, the reward obtained by B may

be decreased. However, since rewards are non-negative, the amount of loss for B is at most $r_B^{j_B}$. By the design of Greedy Maximizer, we have $r_A^{j_A} d_A(k) \geq r_B^{j_B} d_B(k)$ and thus $OPT^{i+1} \geq OPT^i - r_A^{j_A} d_A(k)$.

In sum, for all i, if the Greedy Maximizer schedules (A, j_A) at time slot $i+1$, we have $OPT^{i+1} \geq OPT^i - r_A^{j_A} d_A(k)$. Thus, $GM = OPT^{(k+1)T} \geq OPT^{kT} - GM = OPT - GM$ and $GM \geq OPT/2$. ∎

Finally, we discuss the computation overhead of the Greedy Maximizer. We implement the Greedy Maximizer using a max heap where there is one node for each task X and the value for the node is $r_X^{j_X+1} d_X$. At the beginning of a new frame, that is, in steps 7–11 of Algorithm 2, the values for all nodes are updated and we need to reconstruct the max heap, which takes $O(|S| \log |S|)$ time. At the beginning of a period of task X (steps 12–14), we need to update the value for the node representing X. We can do this by first increasing the value to infinity and making it become the root node of the heap. We then remove the root from the heap, update its value, and insert this node back into the heap. These operations take a total of $O(\log |S|)$ time. Finally, in each time slot, the scheduling decisions (steps 15–17) involve finding the root of the max heap, removing the root from the heap, updating the value of the removed node, and inserting it back into the heap. These operations also take a total of $O(\log |S|)$ time. Thus, the Greedy Maximizer can be efficiently implemented.

VII. Simulation Results

In this section, we present the simulation results. We consider the scenario of a server providing several video streams. For each video stream, there are 12 frames in a GOF, 1 I-frame, 3 P-frames, and 8 B-frames, as suggested in [12]. The server can serve one frame in each time slot.

We first consider a system where all streams have the same frame rate and they all generate one GOF every 30 time slots. We assume that there are two groups of streams, A and B, and each group has 3 different streams. Streams in group A have a higher quality demand, and require that both the I-frames and P-frames are treated as mandatory parts, thus only allowing B-frames to be treated as optional parts. Streams in group B only require I-frames to be mandatory and allow both P-frames and B-frames to be optional. We consider three types of rewards, exponential, logarithmic, and linear. Suppose the optional parts of the k^{th} stream in either group A or B are executed i times in a period. The stream gains a reward of $(5+k)(1-e^{-i/5})$ if the type of reward is exponential, $(5+k)\log(10i+1)$ if it is logarithmic, and $(5+k)i$ if it is linear.

We compare the set of reward requirements of streams that can be fulfilled by the Greedy Maximizer against the set of all feasible requirements. We also compare the policy introduced in [5], which aims to maximize the total per period reward, $\sum_{X \in S} q_X / \frac{T}{\tau_X}$, and thus we call it the MAX policy. To better illustrate the results, we assume that the per period reward requirements for streams in group A are all α and those for streams in group B are all β. We then find all pairs of (α, β) so that the resulting requirements are fulfilled by the evaluated policies and plot the boundaries of all such pairs. We call all pairs of (α, β) that are fulfilled by a policy as the *achievable region* of the policy. We also call the set of all feasible pairs of (α, β) as the *feasible region*.

In each simulation of the Greedy Maximizer, we initiate the debt of X to be $M+1$, where we use the Big-M notation introduced in Section III-A, and run the simulation for 20 frames to ensure that it has converged. We then continue to run the simulation for 5000 additional frames. The system is considered fulfilled by the Greedy Maximizer if none of the mandatory parts miss their deadlines in the 5000 frames, and the total reward obtained by each task is at least 0.995 times its requirement.

The simulation results are shown in Fig. 6. For both the cases of exponential and logarithmic functions, the achievable regions of the MAX policy are rectangles. That is because the MAX policy only aims at maximizing the total per-period rewards and does not allow any tradeoff between rewards of different tasks. The achievable regions of the MAX policy are also much smaller than the feasible regions. On the other hand, the achievable regions of the Greedy Maximizer are the same as the feasible regions for both the cases of exponential and logarithmic functions. This shows that the Greedy Maximizer is optimal when all tasks have the same period.

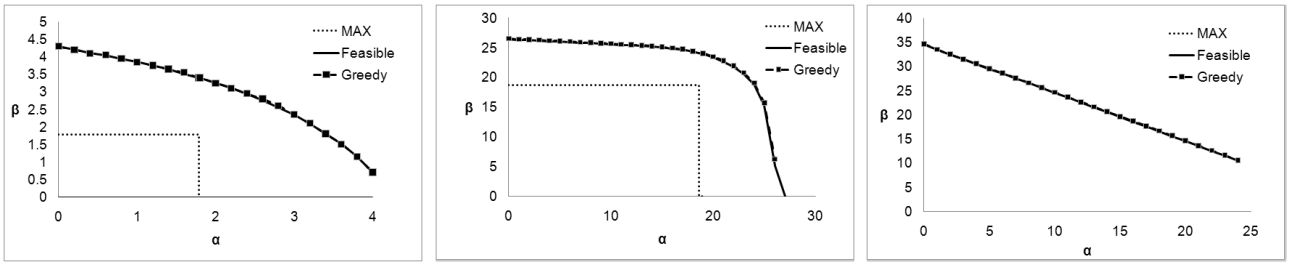
In the linear functions case, the MAX policy fails to fulfill any pairs of (α, β) except $(0, 0)$. A closer examination of the simulation result shows that, besides the mandatory parts, the MAX policy only schedules optional parts of the third tasks in groups A and B. This example shows that, in addition to restricted achievable regions, the MAX policy can also be extremely unfair. Thus, the MAX policy is not desirable when fairness is necessary. On the other hand, the achievable region of the Greedy Maximizer is almost the same as the feasible region.

Next, we simulate a system in which different streams may have different frame rates. We consider the same six streams as in the previous scenarios. The mandatory parts, optional parts, and rewards are the same as in the previous scenario. The only difference is that we assume that the first streams in both group A and B generate one GOF every 40 time slots, the second streams generate one GOF every 30 time slots, and the third streams generate one GOF every 20 time slots.

The simulation results are shown in Fig. 7. As in the previous simulations, the achievable regions of the Greedy Maximizer are always larger than those of the MAX policy, for all functions. Further, the achievable regions of the Greedy Maximizer are very close to the feasible regions. This suggests that, although we have only proved that the Greedy Maximizer is a 2-approximation policy, this approximation bound is indeed very pessimistic. In most cases, the performance of the Greedy Maximizer is close to a feasibility optimal one.

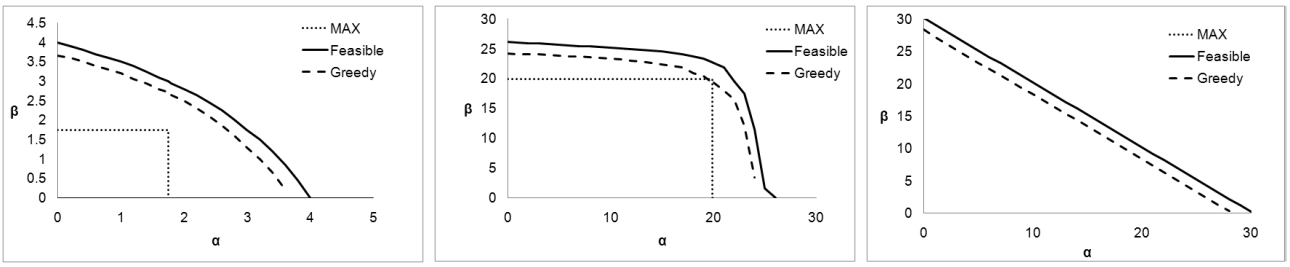
VIII. Concluding Remarks

We have studied a model in which a system consists of several periodic real-time tasks that have their individual



(a) Exponential functions (b) Logarithmic functions (c) Linear functions

Fig. 6: Achievable regions of scheduling policies for the system where all streams have the same frame rate.



(a) Exponential functions (b) Logarithmic functions (c) Linear functions

Fig. 7: Achievable regions of scheduling policies for the system where different streams have different frame rates.

reward requirements. This model is compatible with both the imprecise computation models and IRIS models. By making each task specify its own reward requirement, our model can offer better fairness, and it allows tradeoff between tasks. This feature is important for applications like video stream servers, where it is vital to satisfy the individual requirement for each task. Under this model, we have proved a necessary and sufficient condition for feasibility, and designed a linear time algorithm for verifying feasibility. We have also studied the problem of designing on-line scheduling policies and obtained a sufficient condition for a policy to be feasibility optimal, or to achieve an approximation bound. We have then proposed a simple on-line scheduling policy. We have also analyzed the performance of the on-line scheduling policy and proved that it fulfills all feasible systems in which the periods of all tasks are the same. For general systems where periods may be different for different tasks, we have proved that the on-line policy is a 2-approximation policy. We have also conducted simulations and compared our on-line policy against a policy that maximizes the total reward in the system. Simulation results show that the on-line policy has much larger achievable regions than the compared policy.

Acknowledgement

The authors are grateful to Prof. Marco Caccamo for introducing us to this line of work.

References

[1] J.-T. Chung, J. W. S. Liu, and K.-J. Lin, "Scheduling periodic jobs that allow imprecise results," *IEEE Transactions on Computers*, vol. 39, pp. 1156–1174, September 1990.

[2] J. W. Liu, K.-J. Lin, W.-K. Shih, and A. C.-S. Yu, "Algorithms for scheduling imprecise computations," *Computers*, vol. 24, pp. 58–68, May 1991.

[3] 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*, vol. 45, pp. 802–813, July 1996.

[4] P. Mejia-Alvarez, R. Melhem, and D. Mosse, "An incremental approach to scheduling during overloads in real-time systems," in *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pp. 283–293, 2000.

[5] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Optimal reward-based scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 50, pp. 111–130, February 2001.

[6] H. Cam, "An on-line scheduling policy for IRIS real-time composite tasks," *The Journal of Systems and Software*, vol. 52, pp. 25–32, 2000.

[7] M. Amirijoo, J. Hansson, and S. H. Son, "Specification and management of QoS in real-time databases supporting imprecise computations," *IEEE Transactions on Computers*, vol. 55, pp. 304–319, March 2006.

[8] W.-K. Shih and J. W. Liu, "Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error," *IEEE Transactions on Computers*, vol. 44, pp. 466–471, March 1995.

[9] P. H. Feiler and J. J. Walker, "Adaptive feedback scheduling of incremental and design-to-time tasks," in *Proceedings of the 23rd International Conference on Software Engineering*, pp. 318–326, 2001.

[10] J.-M. Chen, W.-C. Lu, W.-K. Shih, and M.-C. Tang, "Imprecise computations with deferred optional tasks," *Journal of Information Science and Enginnering*, vol. 25, no. 1, pp. 185–200, 2009.

[11] M. Zu and A. M. K. Chang, "Real-time scheduling of hierarchical reward-based tasks," in *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 2–9, 2003.

[12] Axis Communications, "H.264 video compression standard,"

[13] B. Korte and J. Vygen, *Combinatorial Optimization, Theory and Algorithms*. Springer-Verlag Berlin Heifelberg, 2008.