ON THE COMPETITIVENESS OF ONLINE REAL-TIME SCHEDULING WITH RATE OF PROGRESS GUARANTEES

MICHAEL A. PALIS

Department of Computer Science, Rutgers University, Camden, NJ 08102
palis@crab.rutgers.edu

Received 13 October 2002
Accepted 20 March 2003
Communicated by S. Olariu

ABSTRACT

This paper investigates the task scheduling problem in the context of reservation-based real-time systems that provide quality of service (QoS) guarantees. In such a system, each incoming task specifies a rate of progress requirement on the task's execution that must be met by the system in order for the computation to be deemed usable. A new metric, called granularity, is introduced that quantifies both the maximum slowdown and the variance in execution rate that the task allows. This metric generalizes the stretch metric used in recent research on task scheduling. An online preemptive scheduling algorithm is presented that achieves a competitive ratio of \( g(1 - r) \) for every set of tasks with maximum rate \( r \) and minimum granularity \( g \). This result generalizes a previous result based on the stretch metric that showed that a competitive ratio of \( (1 - r) \) is achievable for the case when \( g = 1 \).

1. Introduction

The need to support applications with real-time characteristics, such as speech processing, video-on-demand, and multimedia, has motivated research in operating system frameworks that provide quality of service (QoS) guarantees for real-time applications that run concurrently with traditional non-real-time workloads [1, 6, 7, 10, 11, 12, 13, 14, 18, 19, 20, 22, 23, 24, 25]). In such a framework, a real-time application negotiates with the resource manager a range of “operating levels” at which the application can run depending on the availability of resources. Based on the current state of the system, the resource manager may increase or decrease the application’s operating level within this pre-negotiated range.

One way to characterize an application’s “operating level” is by its stretch factor, which is the amount of slowdown that it can tolerate relative to the time it takes to run on an unloaded system. For example, an application which has an execution time of 100 msec (on an unloaded system) and stretch factor 2 may be allowed to run at a slower rate, or have its execution delayed, so long as it is completed no later than 200 msec after it first becomes runnable. A number of recent papers have addressed the task scheduling problem based on this model [2, 3, 4, 5, 8, 9, 15, 16, 21]. In [4, 5, 21] Muthukrishnan et al. presented offline and online scheduling algorithms.
that minimize either the maximum stretch factor or the average stretch factor among all the tasks in the system. In [8, 9] DasGupta and Palis considered the problem of online task scheduling with admission control, in which each arriving task specifies a maximum stretch factor that it can tolerate and the system admits a task only if it can guarantee that the task can be completed within the specified stretch factor.

The work by DasGupta and Palis appears to be more relevant within the context of reservation-based real-time systems that provide QoS guarantees. However, the notion of stretch factor investigated in their work does not quite capture the rate of progress requirement inherent in many real-time applications, such as the processing of continuous media (e.g., audio and video). In such an application, while it is acceptable to slow down the processing (up to some minimum “operating level”), it is nonetheless crucial that the processing progresses at a relatively uniform rate throughout the lifetime of the application. Unfortunately, the scheduling algorithms described in [8, 9] as well as in [4, 5, 21] are prone to producing “bursty” schedules with non-uniform processing rates, and hence are not applicable to these types of applications.

In [25], Yau and Lam presented a framework for adaptive rate-controlled scheduling that takes into account the rate of progress requirement of real-time applications that we alluded to above. In their framework, the system allows tasks to reserve CPU time based on a rate \( r \) \((0 < r \leq 1)\) and a period \( p \). Specifically, a task is allowed to run for at least \( k r p \) time units over the time interval \( k p \) for \( k = 1, 2, \ldots \), starting from the time the task first becomes runnable. For example, if the rate is 0.2 and the period is 100 msec, then the task is allowed to run for at least 20 ms over the first 100 msec after it arrives, for at least 40 msec over the first 200 msec, and so on. In contrast to Liu and Layland’s periodic task model [17], Yau and Lam’s model allows for the integrated scheduling of both periodic and non-periodic tasks. For example, the task described above may be a numerical application that could run continuously on a lightly loaded system for 5 seconds. If the system becomes heavily loaded, then the system will still ensure that the task runs at least 20k msec for every 100k msec after it arrives, so that it terminates after at most 25 seconds. In [25], Yau and Lam developed an online preemptive scheduling algorithm for their model and presented an empirical evaluation of the algorithm on various applications.

In this paper, we investigate the theoretical performance of online preemptive scheduling algorithms based on Yau and Lam’s framework. Specifically, we consider the problem of scheduling a set of \( n \) independent tasks, \( T = \{ T_1, T_2, \ldots, T_n \} \) on a single CPU system in which each task \( T_i \) is characterized by the following four parameters:

- its arrival time \( a_i \);
- its execution time \( e_i \);
- its rate \( r_i, 0 < r_i \leq 1; \) and
• its period \( p_i \).

The arrival time is the time that the task first becomes runnable. The execution time is the amount of CPU time the task needs to run without interruption on an unloaded system. The last two parameters specify the rate of progress requirement for executing the task. Specifically, upon its arrival at time \( a_i \), task \( T_i \) must be executed for at least \( kr_i p_i \) time units over the time interval \([a_i, a_i + kp_i]\) for \( k = 1, 2, \ldots \) until it is completed (i.e., has run a total of \( e_i \) time units). We assume that \( e_i \geq r_i p_i \).

Define \( g_i \equiv r_i p_i / e_i \) as the granularity of task \( T_i \). Note that \( 0 < g_i \leq 1 \) since \( e_i \geq r_i p_i \). Intuitively, the granularity is a measure of the degree of "smoothness" of the task's execution profile: the "finer" the granularity (i.e., the smaller \( g_i \) is), the more uniform the task's rate of progress will be for any given time interval during its execution. Indeed, when \( g_i = 1 \) for all tasks \( T_i \), the resulting problem instance corresponds to the case when each task \( J_i \) is characterized by a stretch factor equal to \( 1/r_i \). Thus, the granularity metric can be viewed as a generalization of the stretch metric used in [4, 5, 21, 8, 9].

We consider online preemptive scheduling algorithms with admission control that satisfy the following property: Upon the arrival of each task \( T \), the system immediately decides whether to admit or reject \( T \). If it admits \( T \) then the system must execute \( T \) so that it satisfies its rate of progress requirement, as described above. Thus, we assume a hard real-time model in which the system ensures that every admitted task will meet its requirement or the system will be considered to have failed. This is in contrast to a soft real-time system [19] or a firm real-time system [3] that allows the system to fail to meet the requirements of certain admitted tasks, but no "profit" is gained by the system for tasks that do not meet their requirements.

We are interested in online scheduling algorithms that maximize the CPU utilization (or simply utilization), which is defined as the sum of the execution times of all admitted tasks. We measure the performance of an online algorithm \( A \) in terms of its competitive ratio, which is the ratio of the utilization obtained by algorithm \( A \) to that of an optimal offline algorithm \( OPT \). More precisely, let \( U_A(T) \) and \( U_{OPT}(T) \) be the utilizations of \( A \) and \( OPT \), respectively, for the set of tasks \( T \). We say that \( A \) has competitive ratio \( \rho \), \( 0 \leq \rho \leq 1 \), if and only if

\[
\frac{U_A(T)}{U_{OPT}(T)} \geq \rho
\]

for every task set \( T \).

For a given task set \( T = \{ T_1, T_2, \ldots, T_n \} \), let \( r(T) = \max\{ r_i \mid 1 \leq i \leq n \} \) be the maximum rate among all tasks and let \( g(T) = \min\{ g_i \mid 1 \leq i \leq n \} \) be the minimum granularity among all tasks. As discussed earlier, the case when \( g(T) = 1 \) corresponds to the case when each task \( T_i \) is characterized by a stretch factor \( s_i = 1/r_i \) and \( s(T) = \min\{ s_i \mid 1 \leq i \leq n \} \) is the minimum stretch factor among all tasks. In [8, 9] an online preemptive scheduling algorithm was presented for this problem that achieves a competitive ratio of at least \((1 - 1/s) \) for every
task set $T$ with $s(T) \geq s$. In this paper, we generalize this result by demonstrating an online preemptive scheduling algorithm that achieves a competitive ratio of at least $g(1 - r)$ for every task set $T$ with $r(T) \leq r$ and $g(T) \geq g$. Our result implies that under more stringent rate of progress requirements on the tasks’ execution, a competitive online algorithm is still possible, albeit with a proportionately reduced competitive ratio.

The rest of the paper is organized as follows. Section 2 describes the online scheduling algorithm. Section 3 proves the $g(1 - r)$ lower bound on the competitive ratio of the online algorithm and Section 4 proves that this bound is the best possible for this algorithm. Finally, Section 5 concludes the paper with a discussion of some open problems.

2. The Online Scheduler

The online scheduler $A$ is based on the Earliest Deadline First (EDF) scheduling algorithm. When a task $T$ arrives, $A$ first runs an admission test to check if all previously admitted tasks that have not yet completed, plus task $T$, can be run to completion such that their rate of progress requirements are satisfied.

Let $a$, $e$, $r$ and $p$ be the arrival time, execution time, rate, and period, respectively, of $T$. Then the granularity of $T$ is $g = rp/e$. For the purpose of scheduling, one can view $T$ as a sequence of $m = \lfloor 1/g \rfloor$ jobs such that:

(a) for $1 \leq k \leq m - 1$, the execution time and deadline of the $k$-th job in the sequence are $rp$ and $a + kp$, respectively; and

(b) the execution time and deadline of the $m$-th job are $e - (m - 1)rp$ and $a + e/r$, respectively.

Item (b) above takes into account the fact that $1/g$ may not be an integer. The deadline of task $T$ is the deadline of its last job; i.e., $d_T = a + e/r$.

Clearly, $T$’s rate of progress requirement is satisfied if every job in its job sequence can be executed by its deadline. Note, however, that $A$ admits $T$ only if, in addition, all pending jobs associated with previously admitted tasks can be completed by their deadlines.

The details of the scheduling algorithm are as follows. $A$ maintains a queue of jobs associated with tasks that have been admitted but not yet completed. Each job in the queue contains three pieces of information: (1) the task to which it belongs and its position in the task’s job sequence; (2) its deadline, and (3) its remaining execution time (i.e., its execution time minus the CPU time that it has consumed so far). The jobs in the queue are ordered by nondecreasing deadlines. If the CPU is busy, then it is executing the job at the head of the queue (which has the earliest deadline) and the remaining execution time of this job decreases as it continues to execute. The job is deleted from the queue when its remaining execution time becomes zero, and the job (if any) that becomes the new head of the queue is executed next.

When a new task $T$ arrives, $A$ creates $T$’s job sequence then inserts each job $x$ in the sequence in its proper position in the queue, with its remaining execution
time initially set to its (total) execution time. (If there are jobs in the queue with the same deadline as \( x \) then \( x \) is inserted after these jobs.) Then \( A \) performs the following admission test to determine whether to admit or reject \( T \):

\[
\text{reject} = \text{FALSE}; \\
x = \text{first job in } T\text{’s job sequence}; \\
\text{repeat} \\
\quad s = \text{sum of the remaining execution times of all} \\
\quad \text{jobs in the queue up to and including } x; \\
\quad \text{if } s > d(x) \text{ then reject = TRUE;} \\
\quad \text{else } x = \text{next task in the queue after } x; \\
\text{until (} x = \text{NULL or reject = TRUE)};
\]

That is, reject is set to TRUE if \( T \)’s admission will cause some job in the queue to miss its deadline. In this case, \( T \) is rejected and all its associated jobs are removed from the queue. Otherwise, \( T \) is admitted and all of its associated jobs retain their positions in the queue. If \( T \)’s first job is not at the head of the queue, then \( A \) continues execution of the current job at the head of the queue. However, if \( T \)’s first job is installed at the head of the queue, then the current job is preempted and \( T \)’s first job is executed.

As a final note, it may happen that several tasks may arrive simultaneously; in this case, \( A \) processes these tasks in arbitrary order.

3. Competitive Analysis

We prove that the online scheduler described in the previous section achieves a competitive ratio greater than \( g(1 - r) \) for all task sets with maximum rate \( r \) and minimum granularity \( g \).

Before proceeding further, we first introduce some notation to simplify the equations that will be presented shortly. Let \( S \) be a set of tasks. We use \( \Gamma(S) \) to denote the collection of all jobs associated with tasks in \( S \). Additionally, \( \Gamma(S)_{\leq d} \) (\( \Gamma(S)_{> d} \)) denotes the subset of jobs in \( \Gamma(S) \) whose deadlines are \( \leq d \) (respectively, \( > d \)). We slightly abuse notation by using \( e(S) \) to denote the sum of the execution times of all tasks in \( S \). Likewise, if \( X \) is a set of jobs, then \( e(X) \) denotes the sum of the execution times of all jobs in \( X \). Note the \( e(S) = e(\Gamma(S)) \) for any set of tasks \( S \).

Finally, the following inequality will be used later:

**Fact 1.** \( (a + u)/(a + v) \geq u/v \) whenever \( u \leq v \) and \( a \geq 0 \).

**Theorem 1.** For every set of tasks \( T \) with \( r(T) \leq r \) and \( g(T) \geq g \),

\[
\frac{U_A(T)}{U_{OPT}(T)} > g(1 - r). \tag{1}
\]

**Proof.** Let \( T = \{T_1, T_2, \ldots, T_n\} \) be the set of \( n \) tasks. We wish to compare the schedule of \( A \) with that of an optimal offline scheduler \( OPT \). The schedule of \( A \) produces an execution profile that consists of an alternating sequence of busy and idle intervals. A busy interval corresponds to the case when the CPU is busy executing some job. Each busy interval (except the last) is separated from the next
busy interval by an idle interval, during which the CPU is not executing any job. Observe that every task in $T$ must have arrived during a busy interval. If the task arrived when the CPU was idle, then $A$’s queue should have been empty and $A$ would have immediately admitted and executed this task. Additionally, any task admitted by $A$ is completed (i.e., all of its associated jobs executed) during the same busy interval when it arrived.

Let $B_1, B_2, \ldots, B_l$ be the busy intervals of $A$. The tasks in $T$ can be partitioned into disjoint subsets $S_1, S_2, \ldots, S_l$, where $S_i$ is the subset of tasks that arrive during busy interval $B_i$. Let $S_i^A$ and $S_i^{OPT}$ be the subsets of tasks in $S_i$ admitted by $A$ and $OPT$, respectively. Our goal is to show that:

$$\frac{U_A}{U_{OPT}} = \frac{\sum_{i=1}^{l} e(S_i^A)}{\sum_{i=1}^{l} e(S_i^{OPT})} > g(1 - r)$$  \hspace{1cm} (2)

To prove Equation 2, it suffices to show that for every busy interval $B_i, 1 \leq i \leq l$:

$$\frac{e(S_i^A)}{e(S_i^{OPT})} > g(1 - r)$$  \hspace{1cm} (3)

We now prove Equation 3. Consider any busy interval $B$ and let $S$ be the subset of tasks in $T$ that arrive during $B$. (We drop the subscript $i$ for notational convenience). Let $t$ be the time at which interval $B$ begins; thus, all tasks in $S$ have arrival times $\geq t$. Finally, let $S^A$ and $S^{OPT}$ be the subsets of tasks in $S$ that are admitted by $A$ and $OPT$, respectively.

Let $T$ be a task with the latest deadline among all tasks in $S^{OPT} - S^A$. If there is no such task, then $S^{OPT} \subseteq S^A$ and Equation 3 trivially holds, hence we assume that such a $T$ exists. By the admission test, $A$ rejected $T$ because its admission would have caused some job $x$ (possibly $T$’s job) to miss its deadline $d_x$; i.e.,

$$t + e(\Gamma(S^A) \leq d_x) + e(\Gamma(\{T\}) \leq d_x) > d_x$$  \hspace{1cm} (4)

The term $t$ on the left hand side of the above equation is due to the fact that all tasks in $S^A$ have arrival times $\geq t$ and hence cannot be executed before time $t$. Since $\Gamma(S^A) = \Gamma(S^A)_{\leq d_x} \cup \Gamma(S^A)_{> d_x}$, we get:

$$e(\Gamma(S^A)) > d_x - t - e(\Gamma(\{T\}) \leq d_x) + e(\Gamma(S^A)_{> d_x})$$  \hspace{1cm} (5)

Since $OPT$ must complete all jobs associated with tasks in $S^{OPT}$ by their respective deadlines, it should be the case that $t + e(\Gamma(S^{OPT}) \leq d) \leq d$ for every $d \geq t$. Likewise, $\Gamma(S^{OPT}) = \Gamma(S^{OPT})_{\leq d} \cup \Gamma(S^{OPT})_{> d}$ for every $d \geq t$. Therefore,

$$e(\Gamma(S^{OPT})) \leq d - t + e(\Gamma(S^{OPT})_{> d})$$  \hspace{1cm} (6)

for every $d \geq t$.

The rest of the proof is broken down into two cases: (1) $d_T \leq d_x$ and (2) $d_T > d_x$.

**Case 1.** $d_T \leq d_x$. Then $\Gamma(\{T\})_{\leq d_x}$ consists of all jobs in $T$’s job sequence and $e(\Gamma(\{T\})_{\leq d_x}) = e_T$. Equation 5 thus becomes:
\[ e(\Gamma(S^A)) > d_x - t - e_T + e(\Gamma(S^A)_{>d_x}) \]  
(7)

Setting \( d = d_x \) in Equation 6, we get:

\[ e(\Gamma(S^{OPT})) \leq d_x - t + e(\Gamma(S^{OPT})_{>d_x}) \]  
(8)

But \( d_T \leq d_x \) and hence \( \Gamma(S^{OPT})_{>d_x} \) consists only of jobs associated with tasks admitted by OPT whose deadlines are \( > d_T \). Since \( T \) is the task with the latest deadline admitted by OPT but rejected by \( A \), it follows that \( \Gamma(S^{OPT})_{>d_x} \subseteq \Gamma(S^A)_{>d_x} \). Therefore, Equation 8 becomes:

\[ e(\Gamma(S^{OPT})) \leq d_x - t + e(\Gamma(S^A)_{>d_x}) \]  
(9)

From Equations 7 and 9 we get:

\[ \frac{e(S^A)}{e(S^{OPT})} = \frac{e(\Gamma(S^A))}{e(\Gamma(S^{OPT}))} \]
\[ > \frac{d_x - t - e_T + e(\Gamma(S^A)_{>d_x})}{d_x - t + e(\Gamma(S^A)_{>d_x})} \]
\[ \geq \frac{d_x - t - e_T}{d_x - t} \quad \text{by Fact 1} \]
\[ = 1 - e_T/(d_x - t) \]
\[ \geq 1 - e_T/(d_T - t) \quad \text{since } d_T \leq d_x \]
\[ = 1 - e_T/(a_T - t + e_T/r_T) \quad \text{since } d_T = a_T + e_T/r_T \]
\[ \geq 1 - r_T \quad \text{since } a_T \geq t \]
\[ \geq 1 - r \quad \text{since } r_T \leq r \]
\[ \geq g(1 - r) \quad \text{since } 0 < g \leq 1. \]

**Case 2.** \( d_T > d_x \). Setting \( d = d_T \) in Equation 6, we get:

\[ e(\Gamma(S^{OPT})) \leq d_T - t + e(\Gamma(S^{OPT})_{>d_T}) \]  
(10)

Since \( T \) is the task with the latest deadline that was admitted by OPT but rejected by \( A \), it follows that \( \Gamma(S^{OPT})_{>d_T} \subseteq \Gamma(S^A)_{>d_T} \subseteq \Gamma(S^A)_{>d_x} \) (since \( d_T > d_x \)). Therefore, Equation 10 becomes:

\[ e(\Gamma(S^{OPT})) \leq d_T - t + e(\Gamma(S^A)_{>d_x}) \]  
(11)

From Equations 5 and 11, we get:

\[ \frac{e(\Gamma(S^A))}{e(\Gamma(S^{OPT}))} > \frac{d_x - t - e(\Gamma(T)_{\leq d_x}) + e(\Gamma(S^A)_{>d_x})}{d_T - t + e(\Gamma(S^A)_{>d_x})} \]  
(12)
Suppose that $T$’s job sequence consists of $m$ jobs, where $m = \lceil 1/g \rceil$. Let $y$ be the job of $T$ with the latest deadline $d_y \leq d_x$. (If $x$ is a job of $T$ then $y = x$.) Then $y$ is the $k$-th job in $T$’s job sequence, for some $1 \leq k \leq m - 1$. (Note that $y$ cannot be the $m$-th job of $T$ since $d_T > d_x$.) Therefore, $d_y = a_T + k p_T$ for some $1 \leq k \leq m - 1$. Equation 12 then becomes:

$$\frac{e(\Gamma(S^A))}{e(\Gamma(S^{OPT}))} > \frac{a_T + k p_T - t - e(\Gamma(\{T\}) \leq d_x) + e(\Gamma(S^A) > d_x)}{a_T + e_T / r_T - t + e(\Gamma(S^A) > d_x)}$$

$$\geq \frac{k p_T - e(\Gamma(\{T\}) \leq d_x)}{e_T / r_T}, \text{ by Fact 1} \quad (13)$$

But $e(\Gamma(\{T\}) \leq d_x) = k r_T p_T$ since every job of $T$ (except the $m$-th job) has execution time $r_T p_T$ and there are exactly $k \leq m - 1$ jobs of $T$ with deadlines $\leq d_x$ (the $k$-th job being $y$). We therefore have:

$$\frac{e(S^A)}{e(S^{OPT})} = \frac{e(\Gamma(S^A))}{e(\Gamma(S^{OPT}))}$$

$$> \frac{k p_T - k r_T p_T}{e_T / r_T}$$

$$= \frac{k r_T p_T (1 - r_T)}{e_T}$$

$$\geq \frac{r_T p_T (1 - r_T)}{e_T}$$

$$= g_T (1 - r_T)$$

$$\geq g(1 - r),$$

since $g_T \geq g$ and $r_T \leq r$.

4. Tightness of the Lower Bound on the Competitive Ratio

We show that the lower bound on the competitive ratio of the online scheduler $A$ given by Theorem 1 is tight. Specifically, we prove that:

Theorem 2. For every $0 < r \leq 1$ and every $0 < g \leq 1$, there exists a set of tasks $T$ with $r(T) \leq r$ and $g(T) \geq g$ such that:

$$\frac{U_A(T)}{U^{OPT}(T)} \leq \varepsilon + g \left(1 - \frac{1}{1/r}\right),$$

for any arbitrarily small $\varepsilon > 0$.

Proof. Let $s = \lceil 1/r \rceil$. Consider the set of tasks $T = \{T_0\} \cup \{T_i \mid 1 \leq i \leq s\} \cup \{T_2 \mid 1 \leq i \leq s\}$ defined as follows:

- $T_0$ has arrival time $a_0 = 0$, execution time $e_0 = \varepsilon \leq g^2 / s$, rate $r_0 = 1/s$, and period $p_0 = \varepsilon s$;
• The $T_i$'s are identical tasks, each with arrival time $a_1$ such that $0 < a_1 < \varepsilon$, execution time $e_1 = g/s$, rate $r_1 = 1/s$, and period $p_1 = g^2$; and

• The $T_2$'s are identical tasks, each with arrival $a_2$ such that $0 < a_1 < a_2 < \varepsilon$, execution time $e_2 = 1/s$, rate $r_2 = 1/s$, and period $p_2 = g$.

It is easy to verify that every job in $T$ has rate $1/s \leq r$ and granularity $g$.

When task $T_0$ arrives at time $a_0 = 0$, the online scheduler $A$ admits $T_0$ because it is the only task. Now, consider the arrival of the tasks $T_i^k$ ($1 \leq i \leq s$) at time $a_1 > 0$. We claim that $A$ cannot admit all $s$ tasks because some task will not satisfy its rate of progress requirement. Suppose to the contrary that $A$ admits all $s$ tasks $T_i^k$ in addition to $T_0$. Then $A$ must run every $T_i^k$ for at least $e_1g$ time units by time $a_1 + p_1 = a_1 + g^2$. Moreover, since the deadline of $T_0$ is $d_0 = e_0s \leq g^2 < a_1 + p_1$, $A$ must run $T_0$ for $e_0 = \varepsilon$ time units also by time $a_1 + p_1$. Therefore, it should be the case that:

\[
e_0 + se_1g \leq a_1 + p_1 \iff \varepsilon + g^2 \leq a_1 + g^2 \iff \varepsilon \leq a_1,
\]

which is a contradiction since $a_1 < \varepsilon$. Therefore, $A$ can admit $T_0$ and at most $(s - 1)$ tasks $T_i^k$. (It is easy to verify that $A$ can indeed execute $T_0$ and $(s - 1)$ $T_i^k$'s according to their rate of progress requirements.)

Next, consider the arrival of the tasks $T_2^k$ ($1 \leq i \leq s$) at time $a_2 > a_1$. We claim that $A$ cannot admit any one of these tasks. Suppose to the contrary that $A$ admits one of these tasks, say $T_2^i$, in addition to $T_0$ and $(s - 1)$ $T_i^k$'s. Then $A$ must run $T_2^i$ for at least $e_2g$ time units by time $a_2 + p_2 = a_2 + g$. Moreover, since the deadline of each task $T_i^k$ is $d_1 = a_1 + e_1s = a_1 + g < a_2 + p_2$ and the deadline of $T_0$ is $d_0 < d_1 < a_2 + p_2$, then $A$ must run $T_0$ for $e_0$ time units and each of the $(s - 1)$ $T_i^k$'s for $e_1$ time units also by time $a_2 + p_2$. Therefore, it should be the case that:

\[
e_0 + (s - 1)e_1 + e_2g \leq a_2 + p_2 \iff \varepsilon + (s - 1)g + g/s \leq a_2 + g \iff \varepsilon + g \leq a_2 + g \iff \varepsilon \leq a_2,
\]

which is a contradiction since $a_2 < \varepsilon$.

On the other hand, an optimal algorithm $OPT$ will reject $T_0$ and all the $T_i^k$'s and admit only the $s$ tasks $T_2^i$'s. This is clearly possible since for every period of length $p_2 = g$, these tasks require a total of $se_2g = g = p_2$ time units.

As a result, we have:

\[
\frac{U_A(T)}{U_{OPT}(T)} \leq \frac{e_0 + (s - 1)e_1}{se_2} = \frac{\varepsilon + (s - 1)g/s}{s \cdot (1/s)} = \varepsilon + g(1 - 1/s).
\]
5. Concluding Remarks and Future Work

We have presented an online preemptive scheduling algorithm that achieves a competitive ratio of at least $g(1 - r)$ for every task set with maximum rate $r$ and minimum granularity $g$. This bound is tight in that, for any arbitrarily small $\varepsilon > 0$, there are task sets for which the algorithm attains a competitive ratio of at most $g(1 - r) + \varepsilon$. The obvious open question is whether the algorithm is optimal; that is, whether the upper bound on the competitive ratio holds true for all online schedulers. When $g = 1$, the online scheduler is optimal since in [8, 9] it was shown that for $g = 1$, no online scheduling algorithm can achieve a competitive ratio greater than $(1 - r) + \varepsilon$. Thus, the optimality of the algorithm is open for the case when $g < 1$.

Perhaps surprisingly, there is reason to believe that the online scheduling algorithm $A$ is not optimal for the case when $g < 1$. Note that $A$ is a greedy algorithm: it admits an incoming task so long as it can feasibly schedule this task with previously admitted tasks. Consequently, it is easy to construct a sequence of tasks (as we have done in the proof of Theorem 2) that forces $A$ to admit small tasks early in the sequence and reject larger tasks later in the sequence because the larger tasks can no longer be feasibly scheduled with the smaller tasks admitted earlier. On the other hand, one can devise a non-greedy scheduler that rejects certain tasks, even when these tasks can be feasibly scheduled, to allow future "more profitable" tasks to be admitted. On the other hand, since the scheduler has no knowledge of future requests (including the possibility that no further tasks will arrive), it needs to maintain a large enough pool of executable tasks to ensure that the CPU utilization is not too low. It would be interesting to investigate whether one can design such an online scheduler with a better competitive ratio than the greedy scheduler presented here.

The online scheduling algorithm is based on Yau and Lam's task model, which is but one of several models that have been proposed in the literature for reservation-based real-time systems. Other models include the rate-based execution (RBE) model of Jeffay and Goddard [12], the multiframe task model of Mok [20], the generalized multiframe task model of Baruah et al [1], and the CPU Reservations scheduling framework of Microsoft Research's Rialto operating system [13, 14]. For the first three models, analytical tests for determining the feasibility of scheduling task sets are known and have been used in the design of online schedulers. However, the performance of these online scheduling algorithms, from a competitive analysis perspective, has not been analyzed. CPU Reservations has been evaluated empirically on various applications, but its theoretical performance has not been studied. Part of our future work will be to investigate online scheduling algorithms for these real-time task models that achieve provably good competitive ratios.

6. References


