

Co-Scheduling Variable Execution Time Requirement Real-time Tasks and Non Real-Time Tasks

Abhishek Singh
Computer Science Dept
UNC Chapel Hill, USA
asingh@cs.unc.edu

Kevin Jeffay
Computer Science Dept
UNC Chapel Hill, USA
jeffay@cs.unc.edu

Abstract

By scheduling the non real-time tasks earlier while still meeting deadlines for the real-time tasks, the overall system performance can be improved. In particular, we believe that the variability in the execution time requirements of real-time tasks can be effectively leveraged to reduce response times of non real-time tasks. We propose a novel processor sharing algorithm where the processor share of RT jobs increases with their progress based on the empirical probability distribution of execution times of real-time tasks, to adaptively schedule variable requirement real-time to maximize the minimum expected service rate to non real-time tasks at any instant.

1. Introduction

Variability in the execution time requirement of real-time tasks is common in actual real-time systems. For real-time tasks, meeting deadlines is important. Traditional scheduling algorithms schedule real-time tasks assuming that each job requires the worst case execution time (WCET).

This pessimistic approach is essential for meeting deadlines for real time tasks. But practical systems are composed of tasks with varying timeliness requirements. There may be hard real-time tasks, soft real-time tasks, response-time sensitive tasks, and best-effort tasks. The scheduling of real-time tasks impacts the performance of other tasks in the system. In this work, we focus our attention on systems composed of real-time tasks (hard or soft) and response-time-sensitive tasks. In such systems, the rate at which the non real-time tasks get serviced is determined by the schedule of real-time tasks. Classical scheduling algorithms like Rate Monotonic (RM) and EDF, schedule tasks by prioritizing them on their period or deadline. While real-time (RT) tasks support notion of deadlines, non RT tasks usually do not have a crisp deadline associated with them. In this work,

we focus our attention on the class of non RT tasks whose performance is dependent upon their response times.

By scheduling the non real-time (RT) tasks earlier while still meeting deadlines for real-time tasks, the overall system performance can be improved. In particular, we believe that the variability in the execution time requirements of real-time tasks can be effectively leveraged to reduce response times of non real-time tasks. Making assumptions about the arrival time or execution time of non RT tasks is not practical, hence we propose two measures to quantify performance of a RT scheduling algorithm. The first measure denotes the cumulative allocation to non RT tasks by time t and we represent it as $A(t)$. An algorithm with greater $A(t)$ for any t provides better response times to large execution time requirement non RT tasks. This is because, over a large time interval, algorithm with larger $A(t)$ provides greater average allocation to non RT tasks. But a higher value of $A(t)$ does not guarantee that RT tasks do not block non RT tasks. For example, if a non RT task arrives when the RT tasks are scheduled then the non RT task may have to wait for the RT tasks to finish. Delaying RT tasks may not be an option because it may lead to deadline misses. To model this characteristic of RT tasks blocking non RT tasks, we propose quantifying the expected processor share of RT tasks at any time t . A scheduling algorithm which minimizes the maximum expected processor share of RT tasks at any time t maximizes the minimum expected processor share available to non RT tasks at any time t . Greater expected processor share to non RT tasks at any time t provides more responsive service to non RT tasks, specifically small requirement non RT tasks irrespective of their arrival times.

We propose a novel scheduling algorithm using the probability distribution of execution time requirements of real-time (RT) tasks to improve response times of non RT tasks. The RT tasks are scheduled in EDF order, and the active RT job receives processor share given by a function $s(t)$. This gives $(1 - s(t))$ as the processor share available to non RT tasks as function of time and $A(t)$ represents the cumulative

allocation to the non RT tasks as function of time. We calculate function $s(t)$ that minimizes the maximum expected value of $s(t)$ for the RT tasks using probability distribution of their execution time requirements, which can be obtained through online profiling. We show that the proposed schedule combines the effectiveness of Earliest Deadline as late as possible (EDL by Chetto et. al. [4]) in improving $A(t)$ and the efficiency of GPS [9] by allocating certain processor share to the RT tasks thereby eliminating blocking of non RT tasks by RT tasks, without suffering from the drawbacks of either. In particular, the proposed schedule outperforms GPS for the measure $A(t)$ and outperforms EDL in terms of minimizing maximum expected value of $s(t)$ at any time t . In the following sections we define these notions formally.

This paper is organized as follows. First, we present motivation for our work. Next, we show how the proposed approach can be applied to media decoding task which is a well known variable requirement RT task. We follow this by extended the approach to multiple RT task system. In the next section we formally prove how the proposed scheme performs in comparison to typical scheduling algorithms for RT scheduling. Related work is presented in the next section followed by conclusions.

2 Motivation - Single RT Media Decoding Task System

Consider a system with one variable requirement periodic RT task, and other non-RT tasks. Assume that at any instant at least one non-RT task is active.

A periodic RT task τ is characterized by period P and a new job arrives every P time units. Any job may have a worst case execution time requirement of C , which is referred to as its WCET. Let χ be a random variable denoting the execution time of a job.

For example, for a MPEG decoding task, the period P is 40ms (25 fps), the WCET C is 24ms and the average execution time requirement $E[\chi]$ is 10ms. These values were obtained by counting CPU cycles required to decode the frames in Star Wars movie trailer using *mpeg_play*. As can be seen the execution time requirement for the media decoding task is highly variable, with the worst case requirement C being more than twice the mean requirement of $E[\chi]$.

We look at following scheduling approaches to schedule this RT task.

- Priority - The RT task gets priority over non-RT tasks.
- GPS - This algorithm is derived from Generalized Processor Sharing (GPS) [9]. The RT task gets a constant processor share given by its worst case utilization

- EDL - Earliest Deadline as Late as possible (EDL) [4]. The RT task is delayed as much as possible such that it still finishes by its deadline.

We compare these algorithms using the measures $A(t)$ and $s(t)$.

- $s(t)$ denotes the processor share of a RT task as a function of time
- $A(t)$ is the cumulative allocation to other tasks in the system until time t (in the interval $[0,t]$, assuming RT task arrived at time 0), and is given by $(t - \int_0^t s(t)dt)$.

2.1 Defining $s(t)$

The $s(t)$ function is defined in terms of a function $g(\cdot)$ which represents the processor share for a job of the corresponding RT task as a function of time duration since the job's arrival. So function $g(x)$ is defined for $0 \leq x \leq P$. Also since $g(\cdot)$ represents processor share, its value lies between 0 and 1.

Formally,

Definition

$$s(t) = \begin{cases} g(t - \lfloor \frac{t}{P} \rfloor P) & \text{if task active} \\ 0 & \text{otherwise} \end{cases}$$

Note that for a RT task arriving at time 0 and with period P , $(t - \lfloor \frac{t}{P} P \rfloor)$ represents the time duration since arrival of the job active at time t . Furthermore function $g(\cdot)$ should satisfy the following property,

$$\int_0^P g(x)dx \geq C$$

That is, the job should finish C execution time units on or before its deadline which is P time units after its arrival.

The function $g(t)$ where (t is between 0 and P) for the above mentioned scheduling algorithms can be written as follows. We represent the $g(\cdot)$ function for a particular scheduling algorithm as $g_{\text{algorithm}}$ followed by the scheduling algorithm name to differentiate between various scheduling algorithms. Figure 1 illustrates the $g(\cdot)$ functions for Priority GPS and EDL algorithms as well as the Proposed algorithm. The shaded area represent the allocation to the RT job, while the height of the shaded area represents the actual processor share of the RT job at that time. The curve $g_{\text{Proposed}}(\cdot)$ is unique in the sense that it is continuously varying and in the following sections we see how to determine the shape of curve $g_{\text{Proposed}}(\cdot)$ based upon given optimization criterion.

- Priority

$$g_{\text{Priority}}(t) = 1$$

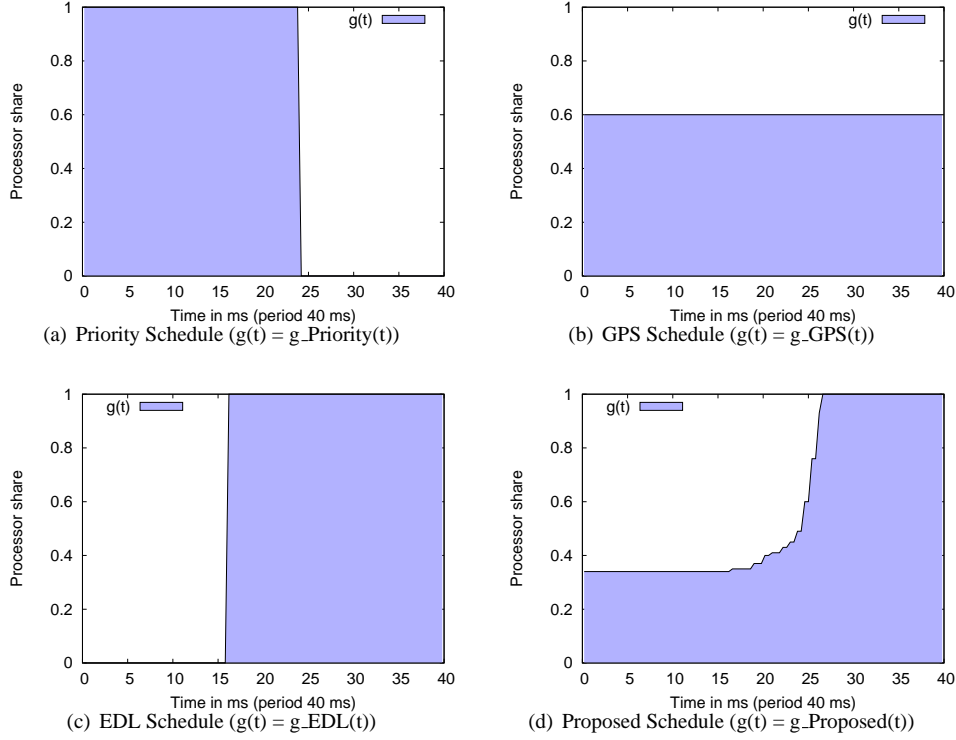


Figure 1. These figures show a job schedule for MPEG task with period 40ms, WCET 24 ms and mean requirement of 10ms. The shaded area represents the allocation to the RT job as a function of the time duration since its arrival. The height of curve $g(t)$ represents the actual processor share of the RT job. The total allocation to the RT job under each of the scheduling policies is equal to the WCET (24ms). While Priority, GPS and EDL $g(t)$ functions are constant, the proposed $g(.)$ function ($g_Proposed(.)$) varies with job progress. In the following sections we describe how $g_Proposed(.)$ is calculated and its properties.

- GPS

$$g_GPS(t) = C/P$$

- EDL

$$g_EDL(t) = \begin{cases} 1 & \text{if } t \geq P - C \\ 0 & \text{otherwise} \end{cases}$$

Lemma 2.1 For any given t between 0 and P ,

$$\int_0^t g_EDL(t)dt \leq \int_0^t g_GPS(t)dt \leq \int_0^t g_Priority(t)dt$$

Proof Under Priority, the RT job is scheduled as soon as it arrives and is given the full processor share ($g_Priority(t)=1$) until it finishes. Under GPS, the RT job is given a constant processor share of $C/P (\leq 1)$ from its arrival until it finishes. Hence $\int_0^t g_GPS(t)dt \leq \int_0^t g_Priority(t)dt$. Under EDL, the RT job starts getting allocation only when the time remaining is just enough to finish the job on deadline under its worst case execution time requirement. Hence, $\int_0^t g_EDL(t)dt \leq \int_0^t g_GPS(t)dt \square$

Lemma 2.2 For any t ,

$$\begin{aligned} \text{Priority } A(t) &\leq \text{GPS } A(t) \\ &\leq \text{EDL } A(t) \end{aligned}$$

Proof By definition, for a single RT media task and a set of non-RT tasks,

$$A(t) = t - \int_0^t s(t)dt$$

Break this integral until the arrival time of latest job (or the deadline of the latest job finished),

$$\begin{aligned} A(t) &= (\lfloor \frac{t}{P} \rfloor P - \int_0^{\lfloor \frac{t}{P} \rfloor P} s(t)dt) \\ &+ ((t - \lfloor \frac{t}{P} \rfloor P) - \int_{\lfloor \frac{t}{P} \rfloor P}^t s(t)dt) \end{aligned}$$

Now the cumulative allocation to the RT task until the arrival time of latest job $\lfloor \frac{t}{P} \rfloor$ is same under any scheduling algorithm where all jobs meet their deadline, hence the first term is same for Priority, GPS and EDL.

The difference is caused by the second term. Now note the second term can be written as

$$((t - \lfloor \frac{t}{P} \rfloor P) - \int_{\lfloor \frac{t}{P} \rfloor P}^t s(t) dt)$$

The negative term $\int_{\lfloor \frac{t}{P} \rfloor P}^t s(t) dt$ represents the allocation to the RT job as function of time. From Lemma 2.1, the allocation to RT job as a function of time since its arrival is greatest under Priority, lesser under GPS and least under EDL. So the cumulative term $((t - \lfloor \frac{t}{P} \rfloor P) - \int_{\lfloor \frac{t}{P} \rfloor P}^t s(t) dt)$ is least under Priority, greater under GPS and greatest under EDL. \square

Lemma 2.3 *For any RT job, let $\max s(\cdot)$ denote the maximum value of $s(t)$.*

$$GPS \max s(\cdot) \leq Priority \max s(\cdot), EDL \max s(\cdot)$$

Proof Note that under GPS, the maximum value of $s(t)$ is $(C/P \leq 1)$, while it is 1 under Priority and EDL. This is because under EDL and Priority scheduling, the RT job is assigned full processor share when it is scheduled. \square

2.2 Understanding $A(t)$, $s(t)$ and $g(\cdot)$

Both $A(t)$ and $s(t)$ are important because while $A(t)$ denotes the cumulative allocation to other tasks in the system in the interval $[0, t]$, $(1 - s(t))$ denotes the instantaneous processor share available to other tasks in the system at time t .

As pointed out earlier, $A(t)$ is same for all scheduling algorithms on job deadlines because the allocation to RT task by any the deadline is same under any scheduling algorithm. The variation is caused when the RT job is active. EDL delays the RT job such that the other tasks in the system get scheduled before the RT task, and hence maximizes $A(t)$. GPS keeps the value of $s(t)$ constant at the worst case utilization of the RT task whenever it is active. Under EDL and Priority, $s(t)$ is 1 while the RT job is active.

$s(t)$ is based on function $g(\cdot)$ which gives the processor share for a RT job as a function of time duration since its arrival (for RT task arriving at time 0 and job arriving every P time units, $t - \lfloor \frac{t}{P} \rfloor P$). The $g(\cdot)$ function is dependent upon the scheduling algorithm used.

2.3 Problem with Priority

If an RT task is given priority over other tasks in the system, then the other tasks are blocked whenever the RT task is active. This gives the worst value of $A(t)$ (minimum) amongst all algorithms guaranteeing that the RT task does not miss any deadline. Also, the value of $(1 - s(t))$ is 0 while the RT task is active and hence this algorithm does not perform well on both measures.

2.4 Problem with EDL

Even though $A(t)$ is maximized by EDL, the value of $(1 - s(t))$ may be 0 while the RT job is active, that is the other tasks arriving when RT job is active are blocked until RT job finishes as in Priority. Furthermore, this blocking time may be arbitrarily long.

For example, consider a RT task with period 1000ms and execution time requirement of 500ms. Let there be non-RT tasks active at any time. In such a scenario, under Priority and EDL, the RT task is active for 500ms, thereby blocking all non-RT tasks for the entire duration of 500ms. The period of the RT task can be chosen arbitrarily long, thereby leading to blocking of non-RT tasks for arbitrarily long intervals.

2.5 Problem with GPS

The advantage of GPS is that the processor share available to other tasks in the system is at least $(1 - C/P)$ at any time. The problem with GPS is that the processor is reserved based on the worst case execution time requirement of the RT task. Thus, even though on average the media decoding task requires just under 10ms of execution time, any job is given a processor share of $24/40 = 0.6$, which is more than twice the mean processor share requirement of $10/40 = 0.25$.

3 Probability and Scheduling

In this section, we start with discussion for a system with single RT task.

As pointed out above, variability in execution time requirement of RT tasks poses unique challenges. First, the scheduling algorithm should provide deadline/allocation guarantees to the RT tasks. Second, to provide guarantees, a non clairvoyant scheduler schedules each job of the RT task assuming that it would require its worst case execution time.

In this section we describe how the variability in execution time requirement can be efficiently handled. We first introduce the notion of expected processor share, $E[s(t)]$. This is the key notion in our analysis. Its importance lies in the fact that for variable requirement RT tasks, the value $s(t)$ is dependent upon whether the RT job has finished or not. If the RT job has finished then it does not require any processor share, but if it is active then it is allocated processor share given by its corresponding $g(\cdot)$ function. So, while the value $s(t)$ at a time t can be thought of as a random variable which may be either 0 or $g(\cdot)$ depending on whether the RT job has finished or not. If the probability distribution of the execution time requirement of the RT

task is known, then the probability that RT job is active after it has finished x units of execution time can be written as $\Pr[\chi > x]$, where χ is the random variable denoting the execution time requirement. Thus, $E[s(t)]$ can be expressed in terms of the $g(\cdot)$ function and the probability distribution of the random variable. Next, we formally define $E[s(t)]$.

Definition $E[s(t)]$ represents the expected value of $s(t)$ for the RT task at time t . Formally,

$$E[s(t)] = g(t - \lfloor \frac{t}{P} \rfloor P) * \Pr[\chi > \int_0^{t - \lfloor \frac{t}{P} \rfloor P} g(x) dx]$$

That is, $E[s(t)]$ at time t is the probability that the RT job is active at time t multiplied by the processor share given by its corresponding $g(\cdot)$ function.

Let $\max E[s(\cdot)]$ denote the maximum expected value of $s(t)$ at any time t .

In this paper, we propose the novel notion of choosing the function $g(\cdot)$ (now we would refer to the $g(\cdot)$ function as $g_Proposed(\cdot)$) for a RT task such that it satisfies the following key properties. For any t , where t is the time duration since arrival of RT job and x lies between 0 and P :

- $\int_0^P g_Proposed(x) dx \geq C$, where C is the WCET
- $g_Proposed(\cdot)$ is chosen such that $\max E[s(\cdot)]$ is minimized

3.1 Calculating $g_Proposed(\cdot)$ for a Single RT Task System

In this section we show how $g_Proposed(\cdot)$ can be calculated which satisfies the above mentioned properties for a system containing single a RT task. At any time some non-RT task is assumed to be active.

Suppose we want to calculate a schedule where $E[s(t)] \leq K$ for any t . Since $E[s(t)]$ is a periodic function with period P , it is sufficient to enforce this relation in the interval $[0, P]$.

For t between 0 and P , $E[s(t)]$ can be written as $g_Proposed(t) \Pr[\chi > \int_0^t g_Proposed(x) dx]$. Thus the constraints can be expressed as,

$$g_Proposed(t) \Pr[\chi > \int_0^t g_Proposed(x) dx] \leq K$$

and,

$$\int_0^P g_Proposed(t) dt \geq C$$

This constraint enforces the fact that any job should finish at most C execution time by its deadline. Such equations also arise in the domain of processor energy savings [7], where the goal is to minimize the expected power consumed.

In this section assume $0 \leq t \leq P$. Now, a schedule with $E[s(t)]$ at most K would do maximum work by any time t if $E[s(t)] = K$ for any t . This is because if $E[s(t)] \leq K$ for some time t , then the function $g_Proposed(t)$ can be increased at that time t thereby leading to greater processor share to the RT task.

The first solution to function $g_Proposed(\cdot)$ can then be written recursively as

$$\begin{aligned} g_Proposed(0) &= K / \Pr[\chi > 0] \\ g_Proposed(t + \delta t) &= \frac{K}{\Pr[\chi > \int_0^t g_Proposed(x) dx]} \end{aligned}$$

But this may lead to $g_Proposed(t)$ outside the range $[0, 1]$ (specifically as the probability approaches 0, $g_Proposed(\cdot)$ approaches infinity). So we limit the value $g_Proposed(\cdot)$ by replacing the RHS in above equation by $\min(1, \frac{K}{\Pr[\chi > \int_0^t g_Proposed(x) dx]})$.

Note that if $g_Proposed(t)$ is reduced the 1 because $\frac{K}{\Pr[\chi > \int_0^t g_Proposed(x) dx]}$ is greater than 1, then $E[s(t)]$ is less than K .

So now the function $g_Proposed(\cdot)$ is defined in terms of a constant K which represents the maximum value of $E[s(t)]$ for this schedule at any time t . What remains is to find the minimum K for which a job of this RT tasks meets its deadline under worst case execution time requirement of C , i.e.

$$\text{minimum } K \text{ s.t. } \int_0^P g_Proposed(t) dt \geq C$$

Now the value of K (the maximum expected processor share) lies between 0 and 1. As the value of K is increased from 0, the execution time allocation to the RT job increases. Therefore a reasonable approximation to K can be efficiently calculated using binary search on the value of K in the interval $[0, 1]$.

3.2 Schedule Illustration - Media Decoding Task Example

Here we present the application of proposed approach on the media decoding task example. The processor cycles used to decode Star Wars trailer using mpeg_play were calculated on a FreeBSD 4.8 machine with 800 MHz Intel Pentium III processor. The worst case execution time requirement was found to be 24ms and the mean execution time requirement was 10ms. To run the movie at 25fps, frames need to be decoded every 40ms. Thus the RT task has a period of 40ms, worst case utilization if $24/40=0.6$ and mean utilization of $10/40=0.25$.

The K calculated for this RT task was 0.34 which is near the mean utilization 0.25 and nearly half the worst case utilization of 0.6 (Refer Fig 2).

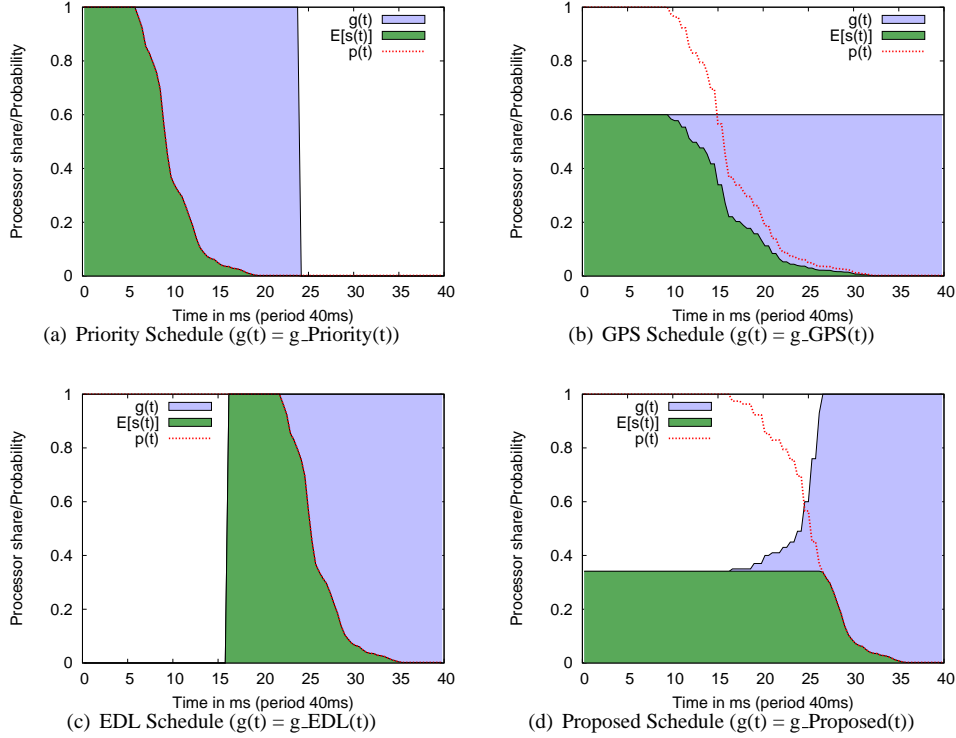


Figure 2. Figure showing job schedules for a MPEG task with period 40ms, WCET 24 ms and mean requirement of 10ms. The height of the shaded region represents the fraction of processor share given to the RT job as a function of the time duration since its arrival. While $g(t)$ curve shows the processor share function corresponding to the scheduling algorithm, $E[s(t)]$ curve shows the expected value of $s(t)$ for a job ($E[s(t)] = g(t)p(t)$ where $p(t) = \Pr[\chi > \int_0^t g(x)dx]$, for t in the range $[0, P]$. χ is the random variable representing job execution time.

3.3 Handling Multiple RT Tasks

The analysis so far considered a system with only a single RT task. In this section we extend the above methodology to a system with multiple RT tasks. But this extension is not trivial. The primary reason being that schedule of one RT task impacts the schedule of other RT tasks. That is, if a job of RT task i is allocated processor share given by function $s_i(t)$, then for a different RT task j , its corresponding processor share function $s_j(t)$ cannot be independent of the value of $s_i(t)$ since the total processor share available to all RT jobs is at most 1 (full utilization).

For the case when all tasks have same period and arrival time, the analysis as done for a single task system can be used, the only difference being the random variable χ will now be $\sum_{i=0}^n \chi_i$ where χ_i is the random variable for the execution requirement of task i and there are n tasks in the system.

If the tasks have different periods then the problem gets tougher. Calculating $s_i(\cdot)$ are dependent because they are constrained by the relation $\sum_{i=1}^n s_i(t) \leq 1$.

Though this is a serious problem, we propose a novel and efficient solution.

Consider a set \mathcal{T} of RT tasks with n tasks. Let task $\tau_i \in \mathcal{T}$ has period P_i , , worst case execution time requirement of C_i and χ_i the random variable representing job execution time.

We form a virtual task $\tau_{virtual}$ with period as 1 time unit, worst case execution time requirement of $U = \sum_{i=1}^n C_i/P_i$, and the random variable representing the execution time requirement of a job as $\chi = \sum_{i=1}^n \chi_i/P_i$.

The schedule minimizing the maximum expected processor share is calculated for this virtual task. Let $g_{virtual}(\cdot)$ be the resulting job function which is defined in the interval $[0, 1]$ and $\int_0^1 g_{virtual}(t)dt = U$.

Now the tasks are scheduled as follows. The function $s_i(t)$ for task τ_i is defined as

$$s_i(t) = \begin{cases} g_{virtual}(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}) & \text{if } \tau_i \text{ active} \\ 0 & \text{otherwise} \end{cases}$$

Note that the function $g_{virtual}(\cdot)$ is such that for a task τ_i with period P_i , the cumulative allocation available to RT tasks while a job of task τ_i is active is $\int_0^{P_i} g_{virtual}(t/P_i)dt = P_i * \int_0^1 g_{virtual}(x)dx = P_i * U$

(note the integral was transformed by the substitution $x * P_i = t$).

At any moment the RT tasks are given a processor share of $s(t) = \max_{1 \leq i \leq n} s_i(t)$, and this processor share is allocated to the earliest deadline active job.

The intuition behind this approach is as follows. Suppose the tasks are given a constant processor share of U when active ($U = \sum_{i=1}^n C_i/P_i$). Then all jobs meet their deadline. And in the worst case a job j may finish exactly on its deadline. In this scenario, the cumulative allocation to RT tasks from the time of arrival of j until its deadline is $U * P_i$ where P_i is the period of task corresponding task for job j . So, in this scenario if the cumulative allocation available to RT tasks from the arrival of job j until its deadline is $U * P_i$, then the job j does not miss its deadline.

Next we formally show that using the proposed scheduling approach all RT jobs meet their deadlines.

Lemma 3.1 Consider a set \mathcal{T} of RT tasks with n tasks. Let task $\tau_i \in \mathcal{T}$ has period P_i , worst case requirement C_i and χ_i be the random variable denoting the execution time requirement of a job. Let $g(\cdot)$ be an increasing function such that $\int_0^1 g(t)dt = U$, where $U = \sum_{i=1}^n C_i/P_i$. The jobs are scheduled using optimal uniprocessor RT scheduling algorithm like preemptive EDF and the processor share at time t is given by $s(t) = \max_{1 \leq i \leq n} s_i(t)$, where

$$s_i(t) = \begin{cases} g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right) & \text{if } \tau_i \text{ active} \\ 0 & \text{otherwise} \end{cases}$$

All jobs meet their deadline.

Proof If the tasks were given a constant processor share of U then all jobs meet their deadline (preemptive EDF schedulability). Suppose all tasks are scheduled in EDF order and the processor share at time t is given by $g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right)$ i.e. $s(t) = g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right)$.

Now note that the function $g(t)$ is such that for a task τ_i with period P_i , the cumulative allocation available for a job is $\int_0^{P_i} g(t/P_i)dt = P_i * \int_0^1 g(x)dx = P_i * U$ (note the integral was transformed by the substitution $x * P_i = t$).

Now if $s(t) = g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right)$, then for any positive integer k , the cumulative allocation available to RT tasks by time $k * P_i$ is $U * k * P_i$, which is same as that available under constant processor share of U . For any t , such that $(k - 1) * P_i \leq t \leq k * P_i$, the cumulative allocation available to RT tasks in the interval $[t, k * P_i]$ is at least $U * (k * P_i - t)$. This is because $g_{Proposed}(\cdot)$ is an increasing function and the cumulative allocation to RT tasks available in the interval $[(k - 1) * P_i, k * P_i]$ is $U * P_i$.

From the point of view of task τ_i , scheduling all RT tasks using $s(t) = g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right)$ is equivalent to scheduling the

RT tasks with a constant processor share of U because by any deadline $k * P_i$ the cumulative allocation available to RT tasks is $k * P_i * U$. Also from any intermediate time t to the nearest next deadline, the cumulative allocation available to RT tasks is U times the time interval. Hence all jobs of τ_i meet their deadline while jobs of other tasks may miss their deadline.

Similarly, if $s(t)$ is chosen as $s(t) = g\left(\frac{t - \lfloor \frac{t}{P_j} \rfloor P_j}{P_j}\right)$ then all jobs of task τ_j meet their deadline while jobs of other task may miss their deadline.

Now suppose $s(t)$ is chosen as $s(t) = \max_{1 \leq i \leq n} g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right)$.

Then, the cumulative allocation by any time t under this schedule is at least equal to the cumulative allocation to RT tasks under a schedule with $s(t) = g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right)$ for any i . Hence jobs of all tasks meet their deadline under this schedule.

Furthermore, if task τ_i is not active at time t (the current job has finished and a new job has not yet arrived), then the function $g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right)$ is redundant since it does not control whether jobs of other tasks would meet their deadlines (which is controlled by the corresponding $g\left(\frac{t - \lfloor \frac{t}{P_j} \rfloor P_j}{P_j}\right)$ term for any other task τ_j). Thus, the term $g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right)$ can be excluded from the $\max_{1 \leq j \leq n} g\left(\frac{t - \lfloor \frac{t}{P_j} \rfloor P_j}{P_j}\right)$. In fact, terms corresponding to all non active tasks can be excluded. Now remember the definition of $s_i(t)$

$$s_i(t) = \begin{cases} g\left(\frac{t - \lfloor \frac{t}{P_i} \rfloor P_i}{P_i}\right) & \text{if } \tau_i \text{ active} \\ 0 & \text{otherwise} \end{cases}$$

Hence the processor share function can be simply written as $s(t) = \max_{1 \leq i \leq n} s_i(t)$. This completes the proof. \square

4 Performance Comparison

In this section we theoretically compare the performance of our algorithm to Priority, GPS and EDL in terms of the measures $A(t)$ and $s(t)$. $A(t)$ represents the cumulative allocation to non-RT tasks by time t and is given by $\int_0^t (1 - s(t))dt$. An algorithm with greater $A(t)$ for any t provides better response time to large non-RT tasks, while an algorithm with lower maximum expected value of $s(t)$ for any t provides better instant service and hence improves responsiveness of shorter non-RT tasks.

The processor share functions for Priority, GPS, EDL and Proposed algorithm for a multiple RT task system are approximated by following $g(\cdot)$ functions. The $g(\cdot)$ function is used to determine the cumulative processor share allocated to RT tasks at any instant, and the RT jobs are scheduled using any optimal uniprocessor RT scheduling

algorithm like preemptive EDF (at processor share $s(t)$). Note that as before, we represent the $g(\cdot)$ function for certain algorithm by prefixing g_{-} before the algorithm name. So $g_{-}Priority(\cdot)$ represents the $g(\cdot)$ function for Priority algorithm.

Note that $0 \leq t \leq 1$, since $g(\cdot)$ functions are assumed to be defined for unit period task. Assume as before we have n RT tasks, where the i^{th} task is represented as τ_i and its period, worst case requirement and execution time requirement are represented as P_i , C_i and χ_i respectively. For feasibility $\sum_{i=1}^n C_i/P_i \leq 1$. Let $U = \sum_{i=1}^n C_i/P_i$. Let $\chi = \sum_{i=1}^n \chi_i/P_i$ which is the random variable representing the combined utilization of all RT tasks.

- Priority, $g_{-}Priority(t) = 1$
- GPS, $g_{-}GPS(t) = U$
- EDL, $g_{-}EDL(t) = \begin{cases} 1 & \text{if } t \geq 1 - U \\ 0 & \text{otherwise} \end{cases}$
- Proposed, $g_{-}Proposed(t) = \min(1, K/p(t))$, where $p(t)$ is the probability that the cumulative RT utilization is greater than $\int_0^t g_{-}Proposed(t)dt$.

Note that for all these algorithms, $\int_0^1 g(t) \geq \sum_{i=1}^n C_i/P_i$ and the $g(\cdot)$'s are increasing functions. So from Lemma 3.1, all these algorithms correctly schedule a given set of RT tasks. Thus all can schedule set of RT tasks.

For the GPS processor share function $g_{-}GPS(t) = U$ is an approximation since processor share of cumulative worst case utilizations of just the active RT tasks is sufficient to correctly schedule the RT tasks. But the simpler formulation is assumed to simplify the proof, though it penalizes GPS in terms of the measure $A(t)$. Note that the maximum expected value of $s_i(t)$ remains unchanged.

For the EDL processor share function, a simplified approximation is again used. This approximation reduces $A(t)$ (the cumulative allocation to non-RT tasks in the system by time t) as compared to the $A(t)$ attainable using the true EDL scheme (as in Chetto and Chetto [4]), where the schedule calculation is done offline for the hyper-period. Despite this simpler EDL formulation, EDL still achieves better $A(t)$ than the the other three algorithms (so the simpler formulation does not skew the results). Again, the maximum expected value of $s_i(t)$ for any task remains unchanged.

Lemma 4.1 For any given t between 0 and 1,

$$\begin{aligned} \int_0^t g_{-}EDL(t)dt &\leq \int_0^t g_{-}Proposed(t)dt \\ &\leq \int_0^t g_{-}GPS(t)dt \leq \int_0^t g_{-}Priority(t)dt \end{aligned}$$

Proof First, note that $\int_0^t g_{-}Proposed(t)dt \leq \int_0^t g_{-}GPS(t)dt$. This is because $g_{-}Proposed(t)$ is a non decreasing function and $g_{-}GPS(t)$ is a constant function. And $\int_0^1 g_{-}Proposed(t)dt = \int_0^1 g_{-}GPS(t)dt = U$. This implies that $\int_0^t g_{-}Proposed(t)dt \leq \int_0^t g_{-}GPS(t)dt$ for $t \leq 1$.

For $\int_0^t g_{-}EDL(t)dt \leq \int_0^t g_{-}Proposed(t)dt$, note that while $g_{-}EDL(t)$ is 0 for t less than $(1-U)$, while $g_{-}Proposed(t)$ is non zero during the interval.

Thus the RT task execution is delayed for the maximum amount in EDL. However, as pointed out earlier, this leads to blocking of non-RT tasks when RT tasks are scheduled. Under the proposed algorithm, RT tasks are delayed lesser than in EDL but their schedule is determined by the execution time requirement probability distribution, thereby reducing blocking of non-RT tasks. \square

Lemma 4.2

$$\begin{aligned} Proposed \max E[s(\cdot)] &\leq GPS \max E[s(\cdot)] \\ &\leq EDL/Priority \max E[s(\cdot)] \end{aligned}$$

Proof The maximum value of $E[s(\cdot)]$ for Priority and EDL is $1 * \Pr[\chi > 0] = \Pr[\chi > 0]$ and for GPS, it is $U * \Pr[\chi > 0]$, where U is the worst case cumulative utilization of RT tasks. This is attained when a RT job begins execution. Also Proposed $\max E[s(\cdot)]$ is K .

What remains to show is that $K \leq U * \Pr[\chi > 0]$. To see this note that, $g_{-}Proposed(0) \leq g_{-}GPS(0)$. This is because while $g_{-}GPS(\cdot)$ is constant and equal to U , $g_{-}Proposed(t)$ is a non decreasing function, so it can start with processor share less than U while still finishing U execution time in a unit sized interval. This gives, $g_{-}Proposed(0) * \Pr[\chi > 0] = K \leq g_{-}GPS(0) * \Pr[\chi > 0]$. \square

Lemma 4.3 The cumulative allocation to non-RT tasks follows the following relation -

$$\begin{aligned} Priority A(t) &\leq GPS A(t) \leq Proposed A(t) \\ &\leq EDL A(t) \end{aligned}$$

Proof Under Priority, the RT tasks get full processor share whenever active and the non-RT tasks are blocked while RT tasks are active, giving the worst value of $A(t)$. Under EDL RT tasks are maximally delayed so EDL has the maximum $A(t)$ value for any t .

What remains to show is the order between GPS and proposed approach. For a single task system, the proposed approach clearly provides larger $A(t)$ for any t . For multiple RT task system, under GPS the RT tasks always get a constant processor share of U whenever active. Under the proposed approach, the RT task may get a smaller value of

processor share during some intervals leading to delayed RT task execution thereby increasing $A(t)$. \square

In summary the proposed scheme delays execution of RT tasks to increase allocation to non-RT tasks by any time t , but at the same time it maintains the maximum expected processor share of RT tasks at any time to be bounded by its minimum value. Thus, under the proposed scheme, larger non-RT tasks get better response times (due to better $A(t)$ than GPS and priority), and smaller non-RT tasks get better response time because of low maximum expected value of $s(t)$ at any time t .

5. Obtaining the Probability Distribution

In the analysis, it was assumed that the probability distribution of execution time requirement is available. Note that the probability distribution does not interfere with the deadline guarantees for the RT tasks because the RT tasks are allocated based on their WCET values.

The function $s(t)$ depends upon the probability distribution. If the estimated probability distribution for execution time requirements is reasonable, then the algorithm performs better in minimizing the maximum expected processor share of RT tasks at any time instant.

The problem of obtaining probability distribution of execution time requirements for real-time tasks has been addressed before [5]. But since for our algorithm, the probability distribution does not impact the deadline guarantees for RT tasks, so even simple online profiling is sufficient for constructing the probability distribution of execution time requirements of the RT tasks.

Basically, we require the distribution $\chi_{virtual} = \sum_{i=1}^n \chi_i/P_i$. This can be obtained by directly profiling the cumulative utilization of RT tasks whenever any RT job finishes. The cumulative utilization is obtained by summing the utilization requirement of the finishing RT job and the utilization requirement of latest finished jobs of other RT tasks in the system. Using these profiled values the histogram for $\chi_{virtual}$ can be constructed. Initially, it is assumed that the $\chi_{virtual}$ is constant and its value is equal to $\sum_{i=1}^n C_i/P_i$, which guarantees that no RT deadlines are ever missed (though the probability that the cumulative utilization of tasks is $\sum_{i=1}^n C_i/P_i$ may become arbitrarily small). The function $s(t)$ is updated periodically (say every 100 new profiled values for $\chi_{virtual}$). With time, the histogram becomes richer and gives a more reasonable approximation to $\chi_{virtual}$.

6. Related Work

Scheduling RT tasks has predominantly focused on meeting deadlines and the variability in execution require-

ment has largely been ignored. In this work, we address the issue of handling variability in execution requirements while meeting deadlines. In particular, we propose the measures $s(t)$ and $A(t)$. $(1 - s(t))$ is the processor share allocated to RT tasks at time t and $A(t)$ is the cumulative allocation to non-RT tasks until time t .

Delaying RT tasks improves allocation $A(t)$ to other tasks in the system, but the RT tasks may require greater allocation later. Earliest Deadline as late as possible (EDL) proposed by Chetto and Chetto [4] is based on this strategy. Dual-priority scheduling algorithm [6] extends the notion of EDL to static priority systems.

Calculating an EDL schedule is non trivial as it requires knowledge of arrival times of all RT tasks beforehand. Algorithms which relax this constraint were proposed by Buttazo et. al. [3]. But the problem with this approach is that while the RT jobs are delayed initially, they are given full processor share later in order to meet deadlines, and any non-RT job arriving during the time when a RT job is given full processor share is delayed until the RT job finishes. In practical systems, the arrival times of non-RT tasks may not be available in advance and the non-RT tasks may have varying degrees of response time sensitiveness. Thus, a scheduling strategy which leads to intervals when non-RT tasks are blocked, may lead to avoidable response time delays of time sensitive non-RT tasks.

Reserving processor share/bandwidth for RT tasks, as in Constant Bandwidth Server (CBS) [1], reduces the measure $A(t)$ as compared to EDL. However the processor share is based on worst case execution time requirement of the RT jobs. In case of variable requirement RT tasks, allocating processor share based on worst case value is not efficient because the WCET may be arbitrarily greater than the mean execution time requirement of the RT jobs. Our approach combines the elegance of share based scheduling with the effectiveness of EDL (delaying RT tasks to improve response times of non-RT tasks). In our approach, the RT tasks are initially assigned lesser processor share than their worst case processor share requirement and their processor share is increased with progress. This leads to a schedule where the RT job may end up requiring the entire processor near its deadline, but the probability that the job actually reaches this phase is low.

In this regard, feedback-control scheduling is a useful approach to continuously adapt the processor share of RT tasks using feedback-control as in [8] and [2]. There are two major problems with this approach. First, some RT jobs may miss deadlines which may or may not be acceptable. In our approach, RT jobs do not miss their deadline. Second, feedback-control assumes dependence between the execution time requirements of job sequences, and leaves it to the system designer to design the appropriate feedback-control function for that dependence. But a job's execu-

tion time requirement may not be dependent on a previous job's requirement. Also, even if such a dependence exists, mathematically qualifying this dependence and designing an appropriate feedback-control loop is difficult. In our approach, the only knowledge required is the execution time distribution of RT tasks which can be easily obtained through online profiling.

7. Conclusion and Discussion

In this work we proposed the novel notion of varying the processor share of the RT tasks with their progress. What this achieves is that the RT job starts off requiring lesser processor share than its worst case utilization, and its processor share increases as the RT job progresses, and so does the probability that the RT job will finish. So while some RT jobs which require execution time requirement near the WCET may end up consuming greater processor share near their deadline, the probability that this scenario arises is less. And the function $s(t)$ is calculated using the probability distribution of execution requirement of RT task obtained through online profiling such that the maximum expected value of $s(t)$ for any time t is minimized.

In this work we assumed the GPS model of processor sharing. The RT jobs get a share $s(t)$, and the non-RT tasks get the remaining processor share $(1 - s(t))$. Most current processors execute tasks sequentially and use a quantum based scheduler to schedule multiple tasks concurrently. Our proposed scheduling algorithm can be adapted to a quantum-based scheduler, and the quantum size would determine the allocation accuracy. Smaller the quantum size better the allocation accuracy.

The prime contribution of our work is the use of probability distribution to schedule variable execution time RT tasks while meeting deadlines. While current scheduling algorithms like Priority, EDL or GPS may perform arbitrarily in terms of response time to non-RT tasks, the proposed scheduling algorithm works well for both measures $A(t)$ and $(1 - s(t))$. Our work opens new doors in the area of scheduling variable requirement RT tasks and we believe that many more exciting applications are possible using this approach. We are currently working on an actual generic prototype implementation of the proposed scheduling algorithm in General Purpose Operating Systems (GPOS). While our approach performs well in terms of the measures $A(t)$ and $(1 - s(t))$, we hope to use this implementation to provide empirical results for non-RT tasks' response times to support our hypothesis.

Acknowledgements

Our sincere thanks to Dr Alan Burns and the reviewers, for their comments on the paper.

References

- [1] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *IEEE Real-Time Systems Symposium*, pages 4–13, 1998.
- [2] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *IEEE Real-Time Systems Symposium*, pages 71–80, 2002.
- [3] G. C. Buttazzo and F. Sensini. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Trans. Computers*, 48(10):1035–1052, 1999.
- [4] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Trans. Software Eng.*, 15(10):1261–1269, 1989.
- [5] L. David and I. Puaut. Static determination of probabilistic execution times. In *ECRTS*, pages 223–230, 2004.
- [6] R. Davis and A. J. Wellings. Dual priority scheduling. In *IEEE Real-Time Systems Symposium*, pages 100–109, 1995.
- [7] J. R. Lorch and A. J. Smith. Pace: A new approach to dynamic voltage scaling. *IEEE Trans. Computers*, 53(7):856–869, 2004.
- [8] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1-2):85–126, 2002.
- [9] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. In *INFOCOM*, pages 521–530, 1993.