Analysis of Latest Defer Time for Fixed-Priority Scheduling Algorithm with Dual Priority

Chiahsun (Alex) Ho and Shelby Funk
Computer Science Department
University of Georgia
Athens, GA, USA
{*ho,shelby*}*@cs.uga.edu*

Abstract

We present an online adjustment method to discover the latest defer time for each job using Fixed Priority scheduling algorithm. By deferring the preemption of lower priority jobs by higher priority jobs until the latest defer time, we reduce the overhead caused by preemptions while still ensuring every job meets its deadline. We present our ongoing work which modifies dual priority of Fixed-Priority scheduling using the worst case response time. Thus we find each task's latest defer time by subtracting its deadline from its worst case response time offline. Then for each job's release time, we adjust the latest defer time online. In this manner, we can then reduce certain amount of preemptions hence reduce the overhead.

1. Introduction

In real-time systems, correct behavior depends not only upon logical correctness, but also upon temporal correctness. In these systems, all jobs have deadlines. In a hard real-time system, each job must be completed no later than its deadline. Even so, there may be times when we might deliberately delay the execution of a job provided it will not cause a deadline to be missed. In this paper, we present an off-line analysis with modified Fixed-Priority scheduling using Dual Priority. This method originally introduced by Davis, et al. [1], calculates a duration during which we can *safely* delay a preemption. We propose an online method of improving their work by adjusting the preemption defer interval as tasks release new jobs.

Preemption occurs when a higher priority task arrives while a lower priority task is executing. In much of the literature, the cost of preemption is assumed to be included within the worst case execution time (WCET) (or sometimes ignored altogether). However, in actual implementations, preemption does incur overhead. These costs occur for two reasons. First, registers of the currently executing (low priority) task must be stored to ensure proper operation upon resumption of execution. Second, the high priority task may cause the contents of the cache to be overwritten, leading to cache misses that would not have occurred without the preemption. Davis, et al., proposed to incorporate deferred preemption into existing scheduling algorithms. The goal is to defer enough that the low priority job finishes executing before the preemption occurs. We propose to extend the duration of the deferred preemption to increase the likelihood that the preemption will no longer be necessary.

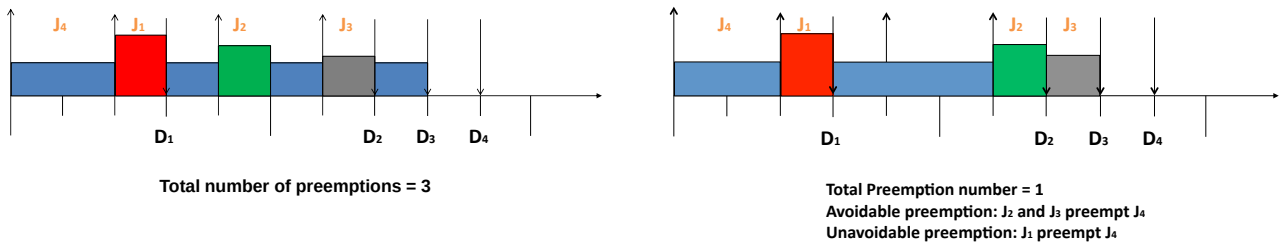
The example below illustrates while some preemptions cannot be avoided without causing deadlines to be missed, others could be avoided.

Example 1. *Consider a system of four jobs J_1, J_2, J_3 and J_4 that arrive at times 2, 4, 6 and 0, respectively, and have deadlines 3, 7, 8 and 9, respectively. J_4 executes for 5 time units and jobs J_1, J_2 and J_3 each execute for 1 time unit. Figure 1 illustrates two schedules for these jobs. Figure 1(a) shows that the preemptive schedule causes 3 preemptions: J_1, J_2 and J_3 each preempt J_4. The first preemption is unavoidable — J_1 must execute as soon as it arrives. However, the preemptions caused by J_2 and J_3 can be avoided without missing any deadlines. Figure. 1(b) illustrates that by deferring those jobs J_2 and J_3, we can reduce the number of preemptions while maintaining feasibility.*

□

In Robert Davis and Andy Wellings [1], they presented the dual priority scheduling algorithm and find the worst case response time by Time Demand Analysis (TDA) [2]. We extend their work to calculate the worst case response time by checking the worst case arrival pattern at each job's release time. Hence we want to make the deferral interval as long as possible but without missing any deadlines.

The remainder of this paper is organized as follows. Section 2 introduces our model and defines important terms to be used in this paper. Section 3 recaps the related work done by Davis et al. Section 4 introduces our extension work for online adjustment. Finally, Section 5 concludes our work and discusses future avenues of research.



(a) Preemptive schedule.

(b) Deferred preemptive schedule.

Figure 1. Two valid schedules.

2. Model and Definitions

We consider a method of reducing preemptions when scheduling periodic [3] or sporadic [4], [5] tasks. Below we introduce the terms and notation we will be using in the following sections.

Periodic and sporadic tasks: The periodic [3] and sporadic [4], [5] task models have proven very useful for the modeling and analysis of real-time computer application systems. In this model, real-time processes recur at regular intervals. Each periodic or sporadic task T_i is characterized by two parameters — a worst case execution requirement (e_i) and a period (p_i). Accordingly, we will model a real-time system $\tau \equiv \{T_1, T_2, \ldots, T_n\}$ as being comprised of a collection of n periodic or sporadic tasks.

In this paper, we will assume that all the system parameters — the number of tasks in the system, the execution requirement and period parameters of each task — are a priori known. Each periodic or sporadic task T_i generates an infinite sequence of jobs $T_{i,0}, T_{i,1}, \ldots, T_{i,k}, \ldots$. We denote the arrival time and deadline of a job $T_{i,k}$ to be $a_{i,k}$ and $d_{i,k}$, respectively. A job does not necessarily start at its arrival time. We let $s_{i,k}$ denote the time when job $T_{i,k}$ first starts to execute. A periodic task $T_i = (p_i, e_i)$ with execution requirement parameter e_i and period parameter p_i generates a job $T_{i,k}$ that arrives at time $a_{i,k} = k \cdot p_i$. Each job $T_{i,k}$ needs to execute for e_i units of time by its deadline of $d_{i,k} = (k+1) \cdot p_i$, for all non-negative integers k. For sporadic tasks, p_i indicates the *minimum* amount of time between two jobs. If a sporadic task generates a job $T_{i,k}$ at time $a_{i,k} = t$, then that job needs to execute for e_i units of time by its deadline of $d_{i,k} = t + p_i$, and T_i's next job $T_{i,k+1}$ can arrive no earlier than $t + p_i$ — i.e, $a_{i,k+1} \geq a_{i,k} + p_i$ for all $k \geq 0$.

For both periodic and sporadic tasks, we denote the worst case response time for each task T_i as r_i.

Latest Defer Time: The latest defer time, Λ_i, for a task τ_i is the latest time τ_i can defer (relative to its arrival time) and ensure it will meet its deadline d_i – i.e., if τ_i defers more than Λ_i, there exists some arrival patterns that will cause τ_i to miss its deadline. We denote each job $T_{i,k}$'s *latest start time* $\lambda_{i,k}$ — thus, $\lambda_{i,k} = a_{i,k} + \Lambda_i$.

We assume a preemptive schedule. Thus, if a job arrives while another job is executing, if the newly arrived job has a higher priority it can interrupt (or preempt) the currently executing job. The costs associated with preemption are generally assumed to be included in the tasks worst case execution times. We let $\boldsymbol{T}_i^{hp}$ denote those tasks whose priority are the same or higher than task T_i, i.e., $\boldsymbol{T}_i^{hp} = \{T_j \mid j \leq i\}$. Similarly, the tasks whose priority are lower than task T_i are denoted $\boldsymbol{T}_i^{lp}$ where $\boldsymbol{T}_i^{lp} = \{T_j \mid j > i\}$.

Deferred preemption intervals: This paper introduces a strategy for reducing the number of preemptions by delaying preemptions when higher priority jobs arrive. We discuss how to incorporate this strategy into Fixed-Priority scheduling algorithm. We define deferral durations as follows.

Let $\tau = T_1, T_2, \ldots, T_n$ be a task set. Assume task T_r is executing and task T_x generates a higher priority job at time t. We define the latest defer time Λ_x as longest deferal preemption interval that lower priority jobs can execute. I.e., the algorithm opts to allow task T_r to continue to executing for an additional Λ_x time units even though T_x has higher priority. When this occurs, we say T_x defers its preemption of T_r, or, alternatively, T_r forces T_x to defer its preemption.

Clearly, we would like to make Λ_x large enough that task T_r is able to finish executing before task T_x has to preempt. Thus, we want the value of Λ_x to be as large as possible but *not* cause any deadlines to be missed.

3. Dual Priority Scheduling

Our goal is to reduce preemptions without sacrificing schedulability. The approach we pursue is to determine the value of Λ_x, a *safe* amount of time to defer preemptions (i.e., a duration that will not cause any deadlines to be missed). We begin by discussing how to find the latest defer time.

We say a job $T_{i,h}$ is in elevated state if $s_{i,h} \geq \lambda_{i,h}$. We first present Fixed-Priority with Dual Priorities introduced by Davis et al. Once $T_{i,k}$ has deferred its preemption by

Λ_i amount of time, it becomes more urgent than it used to be. Hence, we use the *dual priority* scheduling method. When a higher priority task T_x allows a lower priority task to execute for its maximum defer length Λ_x it becomes move urgent that T_x must execute. If this occurs we elevate T_x's priority. In this method, we use an *elevated flag* to indicate if a task has higher priority than its original priority. The tasks in elevated state have higher priority than the tasks in their original *unelevated* state, for instance: T_1 normally has higher priority than T_2. However if T_2 is in elevated state and T_1 is not, then T_2 has higher priority than T_1. If a task has not started to execute before the end of its deferral interval, then its elevated flag is set, giving it higher priority. After the elevated task has completed then the *elevated flag* will set back to false.

In Section 3.5 of [1], Davis et al. have analyzed the latest defer time (which they call the priority promotion time). Let Λ_i be the latest defer time for task τ_i, r_i is the worst case response time and d_i is the deadline of T_i. Then Davis et al. [1] showed $\Lambda_i + r_i = d_i$. Furthermore, there exists a scenario in which τ_i finishes exactly at its deadline when its priority is elevated Λ_i time units after it arrives. The worst case arrival pattern during lower priority task's execution window has also been analyzed by Davis et al. in [1] section 3.2 by using Time Demand Analysis (TDA). With this in mind, let Λ_i be the latest defer time length for τ_i. A critical instant occurs for T_i if during $[a_i, \lambda_{i,k}]$ only tasks in $\boldsymbol{T}_i^{lp}$ or tasks with elevated priority execute and at time $\lambda_{i,k}$ all tasks in $\boldsymbol{T}_i^{hp}$ have their priorities elevated. The following example illustrates the concept of extending deferral time.

Example 2. *Consider a system of four tasks T_1, T_2, T_3 and T_4 shown in Figure 2. The first and the last jobs of T_1 do not execute while T_3 is active. Therefore, we can move T_3s elevation point from 6 to 7, allowing T_4 to complete without being preempted. The worst case response time $r_3 = 14 = \lceil \frac{14}{6} \rceil \cdot 1 + \lceil \frac{14}{10} \rceil \cdot 2 + 7$.*

□

4. Extending the priority elevation point

The TDA equation in [2], $w_i(t) = \sum_{j=1}^{i} c_j \cdot \lceil t/T_j \rceil$ gives the cumulative demands on the processor made by higher priority tasks over [0,t]. If r is the WCRT then $w(r) = r$. Notice that $\lceil \frac{p_i}{p_j} \rceil$ measures the number of times T_i interferes with T_j — i.e., T_i executes while T_j is in the wait queue. By using TDA to determine the worst case response time r_i for each task, we can then find the corresponding Λ_i. TDA certainly shows how many interferences between higher priority tasks and lower priority tasks. However the calculation may be too pessimistic. There exist arrive patterns in which some of the interferences of higher priority tasks do not need to be counted against a task's worst case response time. Specifically, let $[\lambda_{j,k}, d_{j,k}]$ be the interval during which T_j is in elevated state.

We observe that the number of times that T_i interferes with T_j's execution during this interval can be reduced whenever two conditions hold. Let $X_{i,j}$ be the number of times T_i interferes with T_j in the calculation of r_i. If we can be certain that current conditions will result in fewer then $X_{i,j}$ interferences, then we can increase λ_j. If the first and last job of T_i that overlap with $[\lambda_j, d_j]$ execute outside of that interval then we can be *sure* that the response time of $T_{j,k}$ will be no more than $r_j - e_i$. Specifically, let $T_{i,h1}$ and $T_{i,h2}$ be the jobs of T_i that overlap with $\lambda_{j,k}$ and $d_{j.k}$ respectively. If $T_{i,h1}$ executes before $\lambda_{j,k}$ and if $T_{i,h2}$ does not devote its priority before $d_{j,k}$ then $\lambda_{j,k}$ can be increased by e_i. The example below shows the idea by shifting higher priority tasks could reduce the number of interference.

Example 3. *Consider the following example shown in Figure 3, a system of 2 tasks: $T_i = (9, 4)$ and $T_j = (15, 6)$. The number of interferences during [13, 28] as illustrated is only one (because the first and last interferences can be removed), however, by using TDA after taking ceiling the number of interferences are three instead.*

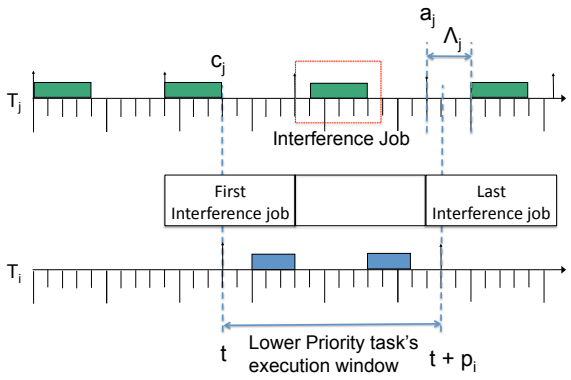
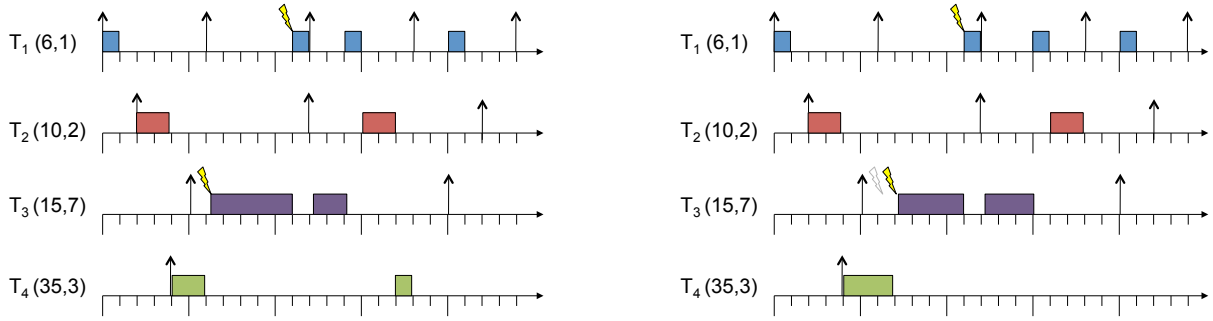


Figure 3. Adjusting response time based on task arrivals

□

Hence we perform an online adjustment in each job's release time for every task. For each higher priority task, if both circumstances above hold we can reduce the number of interference by one. Which allows us to increase the defer time.

To implement this, we add two new arrays-Arr that contain the arrival time of the current job of each task and Comp is a Boolean array which is true for each task whose current job has finished executing. In addition to Λ_i, each task T_i has an associated array $[x_{i,1}, x_{i,2}, ..., x_{i,i-1}]$



(a) Deferral Time without extended

(b) Deferral Time with extended

Figure 2. Extending Deferral Time

containing the number of times the higher priority tasks interfere with T_i when T_i suffers its worst case response time. This is illustrated in Figure 3. and the algorithm is also presented in Algorithm 1. The algorithm is called once at time $\lambda_{i,k}$. Once the defer interval his been extended, Algorithm 1 is not called again.

Algorithm 1 Online Adjustment of $\lambda_{i,k}$

for all higher priority task $T_j, \forall 1 < j \leq i-1$ **do**
  if comp[j] and (Arr[j]+$x_{i.j} \cdot p_j + \Lambda_j \geq d_{i,k}$) **then**
    $\lambda_{i,k} = \lambda_{i,k} + e_j$
  end if
end for

5. Summary and Future Work

In this paper we have presented a method of extending dual priority scheduling with Fixed-Priority scheduling algorithm. When a job's priority is about to be elevated we apply our online adjustment method to increase the defer time. In future we will simulate our online adjustment method to have experimental data for comparing the result with original. In Bril's [6] work, their offline analysis demonstrates that the worst case response time of a higher priority job blocked by lower priority job is based on the amount of time that a task τ_x with a lower priority executes *non-preemptively*. As to our ongoing work is an online extension approach. We would like to extend our work to dynamic-priority scheduling algorithm, such as EDF as well. We also hope to apply this work to systems executing on multiprocessors.

References

[1] R. Davis and A. Wellings, "Dual priority scheduling," in *IEEE Real-Time Systems Symposium*, Dec. 1995, pp. 100–109.

[2] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proceedings of the Real-Time Systems Symposium - 1989.* Santa Monica, California, USA: IEEE Computer Society Press, Dec. 1989, pp. 166–171.

[3] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[4] M. Dertouzos and A. K. Mok, "Multiprocessor scheduling in a hard real-time environment," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1497–1506, 1989.

[5] M. Dertouzos, "Control robotics : the procedural control of physical processors," in *Proceedings of the IFIP Congress*, 1974, pp. 807–813.

[6] R. J. Bril and W. F. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited," in *Proceedings of the 19th Euromicro Conference on Real-Time Systems*. IEEE Computer Society Washington, DC, USA, 2007.