Modifying EDF for Uniform Multiprocessors

Shelby Funk[†]
shelby@cs.unc.edu

Joël Goossens[‡]
Joel.Goossens@ulb.ac.be

Abstract *Scheduling real-time systems on multiprocessors introduces complexities that do not arise when using uniprocessors. If the processors do not all operate at the same speed, scheduling becomes even more complex. When scheduling using* EDF *on multiprocessors, breaking deadline ties in different ways can change the resulting schedule dramatically. We consider methods to resolve the ambiguities in* EDF *priorities due to coincident deadlines. We show that no optimal ambiguity resolver can be both on-line and priority-driven. When processor speeds differ, a job's minimum required execution rate may also be considered in the scheduling decisions. We propose a modification of existing scheduling algorithms that considers processor speed and minimum required execution rate. We show that the resulting algorithm dominates* EDF *on identical multiprocessors and conjecture that it dominates* EDF *on uniform multiprocessors as well.*

1 Introduction

Many people have explored scheduling real-time systems on uniprocessors [3, 4]. However, several complexities arise when the scheduling is performed on a multiprocessor. For example, there are optimal on-line scheduling algorithms for uniprocessors but Dertouzos and Mok [1] showed that no on-line algorithm can be optimal for multiprocessors. The complexity increases if the multiprocessor contains processors that differ from one another. On non-identical multiprocessors the scheduler must determine not only whether a job executes at a given time t, but also which processor the job executes upon. In this paper, we introduce a mechanism that improves on-line scheduling algorithms on non-identical multiprocessors by considering processor speeds when making scheduling decisions. We also introduce a method of resolving ambiguities that arise when using the earliest deadline first (EDF) scheduling algorithm on multiprocessors.

[†]University of North Carolina, Chapel Hill, North Carolina.
[‡]Université Libre de Bruxelles, Brussels, Belgium.

2 Preliminaries

In this section, we introduce the real-time system model and present some definitions regarding real-time instances and multiprocessor platforms.

A **real-time instance** is a collection of time-constrained jobs. Each job j_i is represented by a 3-tuple (a_i, c_i, d_i), where a_i is the job's arrival time, c_i its worst-case execution requirement and d_i its deadline. At any time t, $\xi_i(t)$ denotes the amount of work that has been completed on j_i in the interval $[a_i, t)$. A job is called **active** at time t if $t \geq a_i$ and $\xi_i(t) < c_i$. A **real-time system** is a real-time instance paired with a specific computer processing platform π. A system is called **feasible** if there exists *some* scheduling algorithm for which $\xi_i(d_i) = c_i$ for every job j_i. If an instance I is feasible on a platform π and A is an algorithm that schedules I to meet all its deadlines, then I is called **A-schedulable** on π. A scheduling algorithm A_1 is said to **dominate** another scheduling algorithm A_2 on π if every instance that is A_2-schedulable on π is also A_1-schedulable on π. A_1 **strictly dominates** A_2 on π if A_1 dominates A_2 on π and there is some instance I that is A_1-schedulable but not A_2-schedulable on π. A scheduling algorithm is said to be **priority-driven** if the following holds for every pair of jobs j_i and j_k: if j_i has higher priority than j_k and any point in time, then j_i has higher priority than j_k whenever both jobs are active. An algorithm is **deterministic** if it generates a unique schedule. Finally, an algorithm is **on-line** if at every time t all scheduling decisions are based only upon characteristics of jobs j_i with $a_i \leq t$. In this paper, we are concerned with deterministic on-line scheduling algorithms. Furthermore, the scheduling algorithms must be designed for multiprocessor platforms.

We denote the processors in a multiprocessor platform $p_1, p_2, \ldots, p_m$ where m is the number of processors. An **identical** multiprocessor consists of processors that operate at the same speed. Every job will take the same amount of time to execute on each of the processors. A **uniform** multiprocessor consists of processors that operate at different speeds. When a job executes on a processor of speed s for 1 unit of time, then s units of work are completed. The nota-

tion $\pi = [s_1, s_2, \ldots, s_m]$ denotes a uniform multiprocessor where processor p_i has speed s_i and $s_i \geq s_{i+1}$ for $1 \leq i < m$. The notation π_i denotes the i fastest processors — *i.e.*, $\pi_i \stackrel{\text{def}}{=} \{p_k | k \leq i\}$.

Uniform multiprocessors arise when an identical multiprocessor is upgraded by replacing some, but not all, of the existing processors with faster ones. Also, some multiprocessors may be purchased with processors that execute at different speeds. Note that identical multiprocessors are special cases of uniform multiprocessors — an identical multiprocessor would be denoted $\pi = [1, 1, 1, \ldots, 1]$.

3 Considering Processor Speed

When scheduling on identical multiprocessors, the EDF algorithm is well-defined modulo processor assignment provided all deadlines are unique — *i.e.*, given two valid EDF schedules of the same system, at any time t every job is either executed by both schedules (though not necessarily on the same processor) or not executed by either schedule. However, if the instance has jobs with coincident deadlines or if it is scheduled on a uniform multiprocessor there may be several valid EDF schedules. This occurs because EDF has ambiguities in both these situations.

On identical multiprocessors, a job's performance is not affected by its processor assignment. However, on uniform multiprocessors, the processor assignment can change the resulting schedule dramatically. We use the **greedy** EDF, or G-EDF [2], algorithm on uniform multiprocessors, which assigns higher priority jobs to faster processors. There are still many cases where a feasible instance is not schedulable using G-EDF. Therefore, we wish to consider the speed at which each job must execute in order to complete before its deadline.

Definition 1 ($r_i(t)$) A job's **minimum execution rate** is the ratio of the job's remaining work to the remaining time before its deadline:

$$r_i(t) \stackrel{\text{def}}{=} \frac{c_i - \xi_i(t)}{d_i - t}.$$

This is the minimum average rate at which j_i can execute and still meet its deadline. Therefore, it may be beneficial to consider this value when assigning job priorities. However, if each job's priority is set to r, then preemptions may be unbounded. This is similar to the "thrashing" that occurs while using the least slack first scheduling algorithm. Instead, we use this rate to evaluate a function that can help determine which processors should be used to ensure a job will meet its deadline.

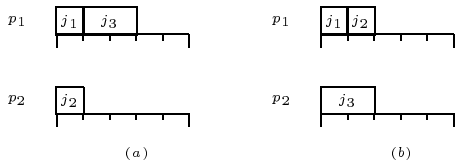

Figure 1: Schedules for the instance I_0 for (*a*) $j_1 > j_2 > j_3$ and (*b*) $j_1 > j_3 > j_2$

Definition 2 ($b(i,t)$) Let I be scheduled on $\pi = [s_1, s_2, \ldots, s_m]$. For each job $j_i \in I$, the **processor blocking function** is the index of the slowest processor that j_i can execute on and meet its deadline:

$$b(i,t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } r(i,t) > s_1, \\ k & \text{if } s_k \geq r(i,t) > s_{k+1} \\ & \qquad (1 \leq k < m), \\ m & \text{if } s_m = r(i,t), \\ m+1 & \text{if } s_m > r(i,t). \end{cases}$$

We now explore ways to modify G-EDF using the minimum execution rate and processor blocking function. In section 4, we prove that ambiguities caused by coincident deadlines cannot be resolved using on-line methods unless a non-priority-driven algorithm is used. We consider using the minimum execution rate to resolve ambiguities. In section 5, we introduce the speed-based mechanism. We prove that using this mechanism cannot cause a feasible instance to become infeasible on identical multiprocessors.

4 Ambiguity Resolvers for EDF

We recently noticed [2] that when using G-EDF on multiprocessors the method chosen to break ties can affect the schedulability of a system. An instance may meet all deadlines using G-EDF, but switching the priority of two jobs with the same deadline could result in deadline misses. Thus, G-EDF is not a deterministic algorithm for instances that contain jobs with the same deadline. In this section, we investigate methods of resolving deadline ties. We call a G-EDF ambiguity resolver **optimal** if the G-EDF schedule using the ambiguity resolver will successfully schedule all G-EDF-schedulable instances. Unfortunately, we will show that no optimal ambiguity resolver exists that is on-line and priority-driven. Therefore, we consider non-priority-driven approaches.

Theorem 1 No optimal on-line priority-driven ambiguity resolver exists for G-EDF.

Proof. Consider the instance $I_0 = \{j_1 = j_2 = (0, 1, 3), j_3 = (0, 2, 3)\}$ and an identical multiprocessor platform composed of 2 processors. There are three possible priority rankings for I_0:(i) $j_1 > j_2 > j_3$, (ii) $j_1 > j_3 > j_2$ and (iii) $j_3 > j_1 > j_2$. (Note these three rankings are equivalent to $j_2 > j_1 > j_3$, $j_2 > j_3 > j_1$ and $j_3 > j_2 > j_1$, respectively.) The G-EDF schedules of I_0 are shown in Figure 1. In these figures, each line represents the activity on a single processor over time. A box on one of the lines indicates that the labelled job executes on that processor for the indicated time interval. Figure (a) illustrates ranking (i) and Figure (b) illustrates rankings (ii) and (iii). Note that in both figures, there are 2 units of time available in the interval $[0, 3)$. In Figure (a), the available time is on a single processor for 2 consecutive time units whereas in Figure (b) the time is simultaneously available on the 2 processors over 1 time unit.

We consider each of the priority rankings separately and show that none of them can be optimal.

(i) Let $I_1 = \{j_3 = j_4 = (2, 1, 3)\}$. $I_0 \bigcup I_1$ is feasible using G-EDF when the priorities of the jobs in I_0 are ranked $j_1 > j_3 > j_2$. However, if $j_1 > j_2 > j_3$, the G-EDF schedule fails. Therefore, if an optimal on-line ambiguity resolver exists, the priority ranking of I_0 can not be $j_1 > j_2 > j_3$.

(ii) Now let $I_2 = \{j_5 = (1, 2, 4), j_6 = j_7 = (3, 1, 4)\}$. $I_0 \bigcup I_2$ is feasible using G-EDF when the priorities of the jobs in I_0 are ranked $j_1 > j_2 > j_3$ but deadlines are missed if they are ranked $j_1 > j_3 > j_2$. Therefore, if an optimal on-line ambiguity resolver exists, the priority ranking of I_0 can not be $j_1 > j_3 > j_2$.

(iii) $I_0 \bigcup I_2$ will also miss deadlines if $j_3 > j_1 > j_2$. Therefore, if an optimal on-line ambiguity resolver exists, the priority ranking of I_0 can not be $j_3 > j_1 > j_2$.

This shows that there is no way to optimally resolve the deadline ambiguities of I_0 using an on-line priority-driven algorithm. □

Theorem 1 demonstrates that we need to sacrifice either the on-line property or the priority-driven property of G-EDF in order to resolve deadline ambiguities in G-EDF optimally on multiprocessors. In many cases, jobs arrive sporadically and therefore we cannot know when future jobs will arrive. Therefore, we want to consider only on-line algorithms. If deadline ambiguities are resolved using the minimum execution rate introduced above, the resulting schedule would no longer be priority-driven for jobs with coincident deadlines. (Note that this is equivalent to using the remaining execution requirement to resolve ambiguities when deadlines are the same.) Resolving ambiguities with the minimum execution rate will feasibly schedule the instances described above. We conjecture that this will give an optimal ambiguity resolver.

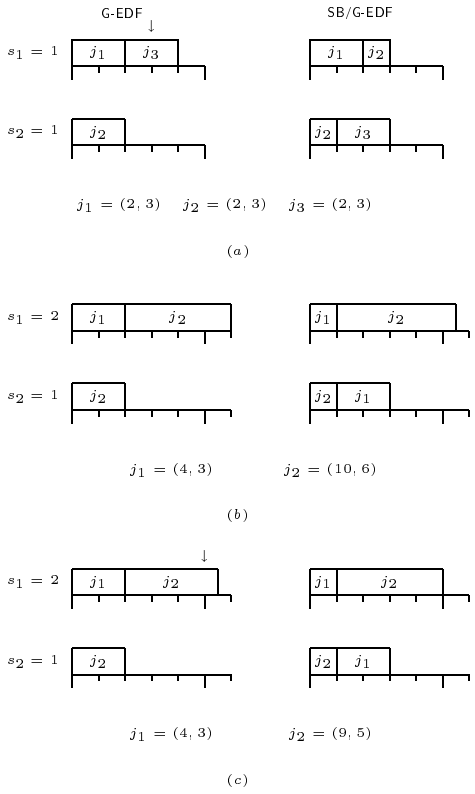

Figure 2: G-EDF and SB/G-EDF schedules

5 Speed-Based Scheduling

In this section, we introduce a mechanism for modifying any algorithm A using the processor blocking function introduced in Section 3. We call this the speed-based, or SB, mechanism. If A is an on-line algorithm, then the modified algorithm will also be on-line. However, the mechanism is not priority-driven.

Definition 3 (SB/A) Let A be any scheduling algorithm. At any time t, consider two active jobs j_i and j_k. The job j_i has higher SB/A priority than j_k if and only if:

- $b(i, t) < b(k, t)$, or
- $b(i, t) = b(k, t)$ and j_i has higher A-priority than j_k.

Figure 2 illustrates G-EDF and SB/G-EDF algorithm for 3 different instances. In Figure 2 (a), the instance is scheduled on a 2-processor identical multiprocessor platform. In Figures 2 (b) and 2 (c), the platforms are not identical. Figures 2 (a) and 2 (c) illustrate systems which are SB/G-EDF-schedulable but are not G-EDF-schedulable (the arrows point to deadline misses). Figure 2 (b) illustrates a system that is schedulable using either G-EDF or SB/G-EDF, but the two algorithms produce different schedules.

Interestingly, though the SB mechanism was designed for uniform multiprocessors, it provides benefits even on identical multiprocessors. When processor speeds are identical, there are three possible b values: $b(i,t) = m+1$ indicates j_i can still make its deadline even if it idles (*i.e.,* j_i has some laxity), $b(i,t) = m$ indicates j_i must execute in order to meet its deadline, and $b(i,t) = 0$ indicates j_i cannot meet its deadline. The lemma below states that even with this coarse granularity, algorithms may be improved by the SB mechanism.

Lemma 2 Let π be any identical multiprocessor and let A be any scheduling algorithm. Then SB/A dominates A on π.

Proof. Let I be any instance that is A-schedulable on π. It suffices to show that I is SB/A-schedulable on π. Since π is identical and I is A-schedulable on π, $b(i,t) \in \{m, m+1\}$ for any job $j_i \in I$ that is active at time t. We consider 2 cases:

1. *For every job $j_i \in I$, $b(i,t) = m+1$ whenever j_i is active.* In this case, the SB/A schedule will be the same as the A schedule since SB/A only alters the priorities of jobs if their b-values differ at some time t.

2. *There exists a job $j_i \in I$ and a time t such that $b(i,t) = m$.* Since I is A-schedulable, j_i must execute in the interval $[t_0, d_i)$, where $t_0 = \min\{t | b(i,t) = m\}$. Since the only possible b-values are m and $m+1$, whenever $b(i,t) = m$ the set of jobs that have higher SB/A-priority must be a subset of those that have higher A-priority. Therefore, the SB/A priority rank of j_i is no lower than the A priority rank and j_i must also be executing in the SB/A schedule.

Thus, in either case, the SB mechanism does not change the the A schedule of I on the identical multiprocessor π. Therefore, I is also SB/A-schedulable on π. □

Theorem 3 SB/G-EDF strictly dominates G-EDF on identical multiprocessors.

Proof. By Lemma 2, SB/G-EDF dominates G-EDF on identical multiprocessors. Moreover, Figure 2 (a) demonstrates that there exists an instance that is SB/G-EDF-schedule, but not G-EDF-schedulable on an identical multiprocessor platform. □

6 Future Research

In this paper, we have considered two possible modifications of G-EDF for uniform multiprocessors. In Section 4 we explored issues that arise when jobs have coincident deadlines. We hope to find an optimal method for resolving such deadline ambiguities. Since we have shown that priority-driven methods cannot be optimal, we would like to consider non-priority-driven methods. In particular, we are hopeful that the minimum execution rate introduced in Section 3 will provide an optimal ambiguity resolver. We would also like to explore the implications of sacrificing the priority-driven property when the deadline-based priorities are ambiguous.

In Section 5, we proved that SB/G-EDF strictly dominates G-EDF on identical multiprocessors. However, the SB/G-EDF algorithm was initially designed to improve EDF using different processor speeds. We would like to prove that SB/G-EDF also strictly dominates G-EDF on uniform multiprocessors. While Figure 2 (c) shows there exists an instance that is SB/G-EDF-schedulable, but not G-EDF-schedulable on some uniform multiprocessor, we have not determined if SB/G-EDF dominates G-EDF on uniform multiprocessors. We also would like to simulate G-EDF and SB/G-EDF and compare the breakdown utilization of the two algorithms for various instances and multiprocessors.

References

[1] Dertouzos, M., and Mok, A. K. Multiprocessor scheduling in a hard real-time environment. *IEEE Transactions on Software Engineering 15*, 12 (1989), 1497–1506.

[2] Goossens, J., Funk, S., and Baruah, S. EDF scheduling on multiprocessor platforms: some (perhaps) counterin tuitive observations. In *Proceedings of the International Conference on Real-time Computing Systems and Applications* (Tokyo, Japan, March 2002).

[3] Liu, C., and Layland, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (1973), 46–61.

[4] Liu, J. W. S. *Real-Time Systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 2000.