Competitive Analysis of Static-Priority Partitioned Scheduling on Uniform Multiprocessors

Björn Andersson

Eduardo Tovar

TR-070802

Version: 1.0

Date: August 2007

Competitive Analysis of Static-Priority Partitioned Scheduling on Uniform Multiprocessors

Björn ANDERSSON, Eduardo TOVAR

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: {bandersson, emt}@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

Consider the problem of scheduling a set of sporadically arriving tasks on a uniform multiprocessor with the goal of meeting deadlines. A processor p has the speed Sp. Tasks can be preempted but they cannot migrate between processors. On each processor, tasks are scheduled with rate-monotonic. We propose an algorithm which can schedule all task sets that any other possible algorithm can schedule assuming that our algorithm is given processors that are $\sqrt{2}/(\sqrt{2}-1) \approx 3.41$ times faster. No such guarantees are previously known for partitioned static-priority scheduling on uniform multiprocessors..

Competitive Analysis of Static-Priority Partitioned Scheduling on Uniform Multiprocessors

Björn Andersson
IPP-HURRAY Research Group
Polytechnic Institute of Porto
Rua Dr. Antonio Bernardino de Almeida 431
4200-072 Porto, Portugal
bandersson@dei.isep.ipp.pt

Eduardo Tovar
IPP-HURRAY Research Group
Polytechnic Institute of Porto
Rua Dr. Antonio Bernardino de Almeida 431
4200-072 Porto, Portugal
emt@dei.isep.ipp.pt

Abstract

Consider the problem of scheduling a set of sporadically arriving tasks on a uniform multiprocessor with the goal of meeting deadlines. A processor p has the speed S_p. Tasks can be preempted but they cannot migrate between processors. On each processor, tasks are scheduled according to rate-monotonic. We propose an algorithm that can schedule all task sets that any other possible algorithm can schedule assuming that our algorithm is given processors that are $\frac{\sqrt{2}}{\sqrt{2}-1} \approx 3.41$ times faster. No such guarantees are previously known for partitioned static-priority scheduling on uniform multiprocessors.

1. Introduction

Consider the problem of preemptive scheduling of a set τ of n sporadically arriving tasks on m processors. A task is given a unique index within the range $1..n$ and a processor is given a unique index within the range $1..m$. The speed of processor p is denoted by S_p, with the interpretation that if a task executes L time units on processor p, it performs $L \times S_p$ units of execution.

A task τ_i generates a (potentially infinite) sequence of jobs. The time when these jobs arrive cannot be controlled by the scheduling algorithm and the time of a job arrival is unknown to the scheduling algorithm before the job arrives. It is assumed that the time between two consecutive arrivals of jobs from the same task τ_i is at least T_i. We say that a job generated by τ_i finishes execution at the time when it has performed C_i units of execution. If a job finishes execution at most T_i time units after its arrival, then we say that the job meets its deadline; otherwise it misses its deadline. It is assumed that $0 < C_i$ and $0 < T_i$, and that T_i and C_i are real numbers. Note that C_i is permitted to be greater than T_i.

The scheduling algorithm is allowed to preempt the execution of a job and there is no cost associated with preemption. Migration is not permitted; when a job resumes execution after being preempted, the job must execute on the same processor as it executed on before it was preempted. Also, if any two jobs are generated by the same task then these two jobs must execute on the same processor. It is assumed that a processor cannot execute two or more jobs simultaneously, and a job cannot execute on two or more processors simultaneously. It is also assumed that T_i and C_i of all tasks are known to the scheduling algorithm.

Our goal is to design an algorithm that schedules tasks to meet the deadlines of all jobs. Unfortunately, the problem of deciding if a set of tasks can be partitioned such that all tasks on each processor meet deadlines is NP-complete [3]. Consequently, the problem of assigning tasks to processors is intractable. For this reason, we will allow an algorithm to fail to assign tasks to processors even when it would be possible to assign tasks to processors such that deadlines would be met. For such scheduling algorithms, it is common to characterize the performance with the notion of a *utilization bound* [13]. This notion has the additional advantage of allowing designers to find out if a specific task set will meet deadlines before run-time; this is often called *schedulability analysis*. Unfortunately, the standard definition of a utilization bound used in uniprocessor scheduling [13] and on multiprocessors with identical processors [1] cannot be applied on uniform processors. For this reason, we will instead use another performance metric: the *speed competitive ratio*.

The speed competitive ratio of an algorithm A is denoted CPT_A. It is the lowest number such that for every task set τ and for every uniform multiprocessor Π', characterized by the speed of processors S'_1,S'_2,. . .,S'_m, it holds that if it is possible (using migration if necessary) to meet all deadlines of τ on Π' then algorithm A meets all deadlines of τ on Π, where Π is a uniform multiprocessor where each processor has a speed CPT_A greater than the corresponding processor in Π'.

A low speed competitive ratio indicates high performance. A speed competitive ratio of 1 is the best achievable. And a speed competitive ratio of two is the best achievable [2] for scheduling algorithms that do not allow migration. If a scheduling algorithm has a finite speed competitive ratio then one can solve every problem instance using processors that are sufficiently fast. If no finite speed competitive ratio has been proven for a scheduling problem then one cannot know if faster processors will ever help.

It is challenging to design a partitioned algorithm with a finite speed competitive ratio. Nonetheless, such an algorithm was designed [2]; this algorithm assumed that Earliest-Deadline-First (EDF) [13] was used on each processor. But most real-time operating systems do not support EDF. Instead, they support static-priority scheduling and here, the priority-assignment scheme Rate-Monotonic (RM) [13] is frequently used. For this reason, there is a need to prove a speed competitive factor for uniform multiprocessors scheduling without migration and using RM on each processor.

Therefore, in this paper we propose a partitioned scheduling algorithm for uniform multiprocessors; it allows preemption and it uses RM [13] on each processor. We prove its speed competitive ratio: it is at most $\frac{\sqrt{2}}{\sqrt{2}-1} \approx 3.41$. For the special case, where the maximum utilization of tasks does not exceed the speed of the slowest processor, the performance is better; we show that the speed competitive ratio is $1+\sqrt{2} \approx 2.41$. Moreover, with simulation of randomly generated task sets, we show that for many task sets our new algorithm needs only a small amount of extra resources (significantly less than 3.41).

The remainder of this paper is organized as follows. Section 2 gives a background on uniform multiprocessors. Section 3 presents our new algorithm which does not migrate tasks and Section 4 proves its speed competitive ratio. Section 5 evaluates the new algorithm through simulation experiments, while Section 6 discusses the ability of previous work to solve the addressed problem. Finally, conclusions are drawn in Section 7.

2. Background

Recall that task migration is not permitted. Therefore, when a job resumes execution after being preempted, the job must execute on the same processor as it executed on before it was preempted. We also assume that if any two jobs are generated by the same task, then these two jobs must execute on the same processor. This type of scheduling is called *partitioned multiprocessor scheduling* because it is equivalent to partitioning the set of tasks such that all tasks in a partition are assigned to its dedicated processor and then a uniprocessor scheduling algorithm is used at run-time. We assume that Rate-Monotonic (RM) is used. It assigns a static-priority to each task; that is, the priority of a task does not change at run-time. A task releases a (possibly infinite) sequence of jobs. A job has the same priority as the task that released the job. At run-time, at every time, the scheduling algorithm selects for execution the job that has the highest priority among the set of tasks that has arrived at that time and still has not finished execution. It is well-known that preemptive Rate-Monotonic (RM) is an optimal static-priority scheduling algorithm on a uniprocessor with our task model; that is, it meets deadlines if there is any static-priority preemptive uniprocessor scheduling algorithm that meets deadlines. For this reason, we will, in the remainder of the paper, assume that preemptive RM is used on each processor. For convenience we will refer to RM with the meaning of preemptive RM.

The problem of partitioning the task set is however non-trivial. It is important that the task assignment algorithm is aware of the scheduling algorithm used on a uniprocessor and it must use a uniprocessor schedulability test to know this. For RM it is known [13] that:

Theorem 1. *Let p be a processor of speed S_p = 1 and let n_p denote the number of tasks assigned to processor p. If $\sum_{i=1}^{n_p} \frac{C_i}{T_i} \leq n_p \times (2^{1/n_p} - 1)$ and tasks are scheduled with RM on p then all deadlines are met.*

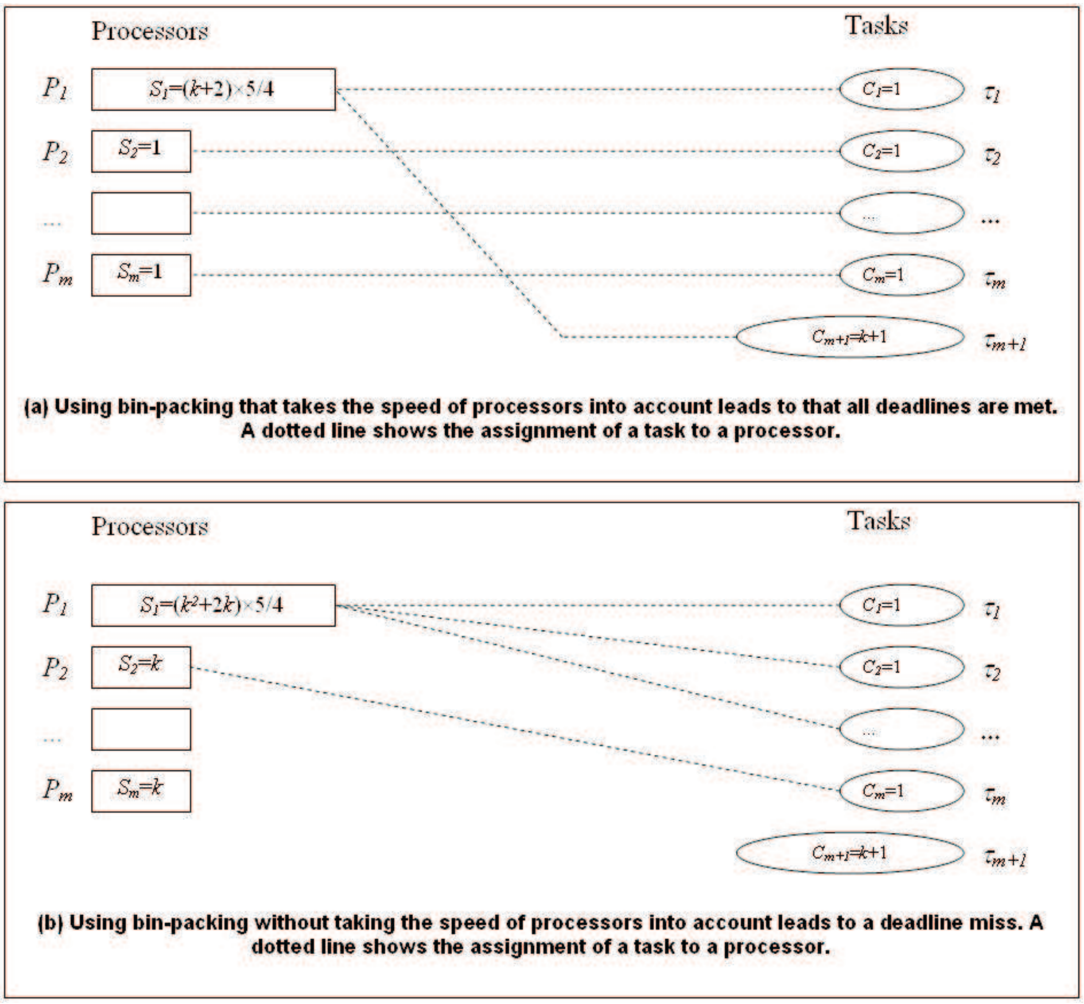
We can easily remove the restriction S_p = 1 from Theorem 1 and phrase Theorem 2 as follows.

Theorem 2. *Let p be a processor of speed S_p and let n_p denote the number of tasks assigned to processor p. If $\sum_{i=1}^{n_p} \frac{C_i}{T_i} \leq S_p \times n_p \cdot (2^{1/n_p} - 1)$ and tasks are scheduled with RM on p then all deadlines are met.*

When assigning tasks to processors, the speed of a processor clearly is used in the schedulability test, for example the one in Theorem 2. But it is also important that processors are considered in the right order, in order to achieve a finite speed competitive ratio. Example 1 illustrates this.

Example 1. *Let k be an arbitrary integer such that $k \geq 3$. Consider n=k^3+1 tasks to be scheduled on m=k^3 processors. All tasks have T_i = 1. Tasks with $i \in$ 1..m, have C_i = 1 and the task m+1 has C_{m+1} = k+1. Processor 1 has the speed S_1 = (k+2) $\times$ 5/4 and the processors with index 2..m have the speed S_p = 1.*

Observe Figure 1. It can be seen (from Figure 1a) that this task set can be scheduled by assigning τ_{m+1} to processor 1 and one of the other tasks to processor 1, and the other tasks given one dedicated processor each. However, consider Figure 1b. If the task assignment scheme considers tasks and processors in order of their index and uses a



(a) Using bin-packing that takes the speed of processors into account leads to that all deadlines are met. A dotted line shows the assignment of a task to a processor.

(b) Using bin-packing without taking the speed of processors into account leads to a deadline miss. A dotted line shows the assignment of a task to a processor.

Figure 1. It is important to exploit knowledge of the speed of the processors when assigning tasks to processors. Otherwise, the speed competitive ratio can approach infinity.

normal bin-packing algorithm, then a deadline is missed. A deadline is still missed even if processors are k times faster. We can see this as follows. Processor 1 will have the speed S_1 = (k^2+2k) × 5/4 and processors 2,3,4,. . .,m will have speed S_p = k. The speed of processor 1 is not enough to host all the tasks 1, 2, 3,. . ., m because their cumulative utilization is k^3 and this exceeds the speed of processor 1, which is S_1 = (k^2+2k) × 5/4 (it is true that $k^3 \geq$ (k^2+2k) × 5/4 since $k \geq 3$). Consequently, task τ_{m+1} will not be assigned to processor 1 and hence τ_{m+1} must be assigned to one of the processors with index 2,3,. . .,m. But τ_{m+1} cannot be assigned to a processor with index 2,3,. . .,m because the utilization of τ_{m+1} is k+1 and the speed of each of the processors is k.

We have seen that algorithms using bin-packing can fail if the speed of the processors is not considered in the assignment algorithm. This can happen although these algorithms are given processors that are k times faster. We can do this reasoning for any $k \geq 3$. By letting $k \to \infty$ we obtain that the speed competitive ratio is infinite for these bin-packing schemes that do not take the speed of each processor into consideration. This stresses the importance of taking the speed of processors into account when the task assignment algorithm makes decisions.

We will now discuss feasibility testing of scheduling with migration permitted; that is, we will state conditions such that if and only if these conditions are true for a task set then it is possible to schedule the task set. These results are useful for proving the speed competitive ratio of the new algorithm in Section 3.

We have showed in [2] by simple reformulation of results in [4] that: A task set is feasible on a uniform multiprocessor platform if and only if $l \leq 1$ for the following optimization problem.

minimize l
subject to:

$$\forall i \in \{1,2,\ldots,n\} : \sum_{p=1}^{m} u_{i,p} = \frac{C_i}{T_i} \tag{1}$$

$$\forall i \in \{1,2,\ldots,n\} : \sum_{p=1}^{m} \frac{u_{i,p}}{S_p} \leq l \tag{2}$$

$$\forall p \in \{1,2,\ldots,m\} : \sum_{i=1}^{n} \frac{u_{i,p}}{S_p} \leq l \tag{3}$$

$$\forall i \in \{1,2,\ldots,n\}, p \in \{1,2,\ldots,m\} : 0 \leq u_{i,p} \tag{4}$$

Intuitively, $u_{i,p}$ in (1)-(4) means the utilization that task τ_i is assigned to processor p.

From (1),(2),(3) and (4) we obtain Lemma 1.

Lemma 1. *If it holds that:*

$$\sum_{p=1}^{m} S_p < \sum_{i=1}^{n} \frac{C_i}{T_i}$$

then no scheduling algorithm can meet all deadlines.

Proof. This lemma was proven in [2]. □

3. The new algorithm

The new algorithm is described in Algorithm 1. It is called RM-DU-IS-FF because it uses RM on each processor, it sorts tasks in order of Decreasing-Utilization, it sorts processors in order of Increasing-Speed and it uses First-Fit bin-packing. Line 12 is the schedulability test from Theorem 2.

It is straightforward to see that the algorithm has the time complexity $O(n \times m + n \log n + m \log m)$. The performance of RM-DU-IS-FF is given by Theorem 3.

Theorem 3. $CPT_{RM-DU-IS-FF} \leq \frac{\sqrt{2}}{\sqrt{2}-1} \approx 3.41$.

We have now seen the speed competitive ratio of RM-DU-IS-FF. But previous work [3] in the real-time scheduling on uniform multiprocessors has focused on the interesting special case where $\forall i,p$ it holds that: $C_i/T_i \leq S_p$. It behooves us to analyze the performance of RM-DU-IS-FF for that case as well. Theorem 4 does that.

Theorem 4. *If we only consider task sets where* $\forall i,p$ *it holds that:* $C_i/T_i \leq S_p$ *then* $CPT_{RM-DU-IS-FF} \leq 1+\sqrt{2}$.

Algorithm 1 RM-DU-IS-FF, a task assignment algorithm for a uniform multiprocessor.

```
1:  sort processors such that $S_1 \leq S_2 \leq \ldots \leq S_m$
2:  sort tasks such that $C_1/T_1 \geq C_2/T_2 \geq \ldots \geq C_n/T_n$
3:  **for** all p in 1..$m$ **do**
4:      U[$p$] := 0
5:      n[$p$] := 0
6:  **end for**
7:  i := 1
8:  **while** (i<=n) **do**
9:      p := 1
10:     allocated := FALSE
11:     **while** (p<=m) **and** (allocated=FALSE) **do**
12:         **if** U[p]+ $C_i/T_i$ <=
13:             $S_p \times (n[p]+1) \times (2^{1/(n[p]+1)}-1)$ **then**
14:             assign task i to processor p
15:             U[p] := U[p]+ $C_i/T_i$
16:             n[p] := n[p]+ 1
17:             allocated := TRUE
18:             i := i + 1
19:         **else**
20:             p := p + 1
21:         **end if**
22:     **end while**
23:     **if** (allocated=FALSE) **then**
24:         declare FAILURE
25:     **end if**
26: **end while**
27: declare SUCCESS
```

We see that the algorithm RM-DU-IS-FF can obtain a tighter bound for such task sets. In fact, Theorem 4 can be seen as a generalization of Oh and Baker's result [14] in real-time scheduling on identical multiprocessors; it offers the same performance bound as the analysis by Oh but for a less restrictive computer platform.

4. Proof of speed competitive ratio

Theorem 3. $CPT_{RM-DU-IS-FF} \leq \frac{\sqrt{2}}{\sqrt{2}-1} \approx 3.41$.

Proof. We will prove the theorem using contradiction. We will do so and show that a failed task set must request more than $\sqrt{2}-1$ of the processing capacity of a subset of processors. We will then consider this task set to be scheduled using a scheduling algorithm where migration is allowed and a computing platform with lower speed is used. It will turn out that every such migrative algorithm must utilize more than the sum of the computing capacity of the subset of processors. This will contradict Lemma 1 and it will prove the theorem. Let us elaborate this reasoning.

If the theorem was false then there exists a task set

```
1:  sort processors such that $S_1 \le S_2 \le \ldots \le S_m$
2:  sort tasks such that $C_1/T_1 \ge C_2/T_2 \ge \ldots \ge C_n/T_n$
3:  for all p in 1..$m$ do
4:      U[$p$] := 0
5:      n[$p$] := 0
6:  end for
7:  i := 1
8:  while (i<=n) do
9:      p := 1
10:     allocated := FALSE
11:     while (p<=m) and (allocated=FALSE) do
12:         if U[p]+ $C_i/T_i$ <=
13:             $S_p \times (n[p]+1) \times (2^{1/(n[p]+1)} - 1)$ then
14:             assign task i to processor p
15:             U[p] := U[p]+ $C_i/T_i$
16:             n[p] := n[p]+ 1
17:             allocated := TRUE
18:             i := i + 1
19:         else
20:             p := p + 1
21:         end if
22:     end while
23:     if (allocated=FALSE) then
24:         declare FAILURE
25:     end if
26: end while
27: declare SUCCESS
```

Figure 2. An algorithm for assigning tasks to processors.

```
1.  for p in 1..$m$ do
2.      U[p] := 0
3.      lo_split[p] := 0
4.      hi_split[p] := 0
5.  end for
6.  Let $\tau^{heavy}$ denote the set of tasks such that $C_i/T_i >$ SEP
7.  Let $\tau^{light}$ denote the set of tasks such that $C_i/T_i \le$ SEP
8.  L := | heavy |
9.  Order tasks such that $\tau_i$ with $i$ in 1..L are all in $\tau^{heavy}$
10.     and $\tau_i$ with $i$ in $L$+1..$n$ are all in $\tau^{light}$
11. Sort tasks in L+1..$m$ such that $T_{L+1} \le T_{L+2} \le \ldots \le T_n$
11. for $i$ in 1..L do
12.     p := i
13.     U[p] := U[p] + $C_i/T_i$
14.     $\tau_i$.processor_id1 := p
15.     $\tau_i$.processor_id2 := p
16. end for
17. p := L + 1
18. for i := L+1 to n do
19.     if U[p]+$C_i/T_i \le$ SEP then
20.         U[p] := U[p] + $C_i/T_i$
21.         $\tau_i$.processor_id1 := p
22.         $\tau_i$.processor_id2 := p
23.     else
24.         if p+1 $\le$ m then
25.             hi_split[p] := SEP-$C_i/T_i$
26.             lo_split[p+1] := $C_i/T_i$-hi_split[p]
27.             $\tau_i$.processor_id1 := p
28.             $\tau_i$.processor_id2 := p+1
29.             $U[p]$ := $U[p]$ + hi_split[p]
30.             $U[p$+1] := $U[p$+1] + lo_split[p+1]
31.             $p$ := $p$+1
32.         else
33.             declare FAILURE
34.         endif
35.     end if
36. end for
37. declare SUCCESS
```

TF such that RM-DU-IS-FF declares FAILURE on multiprocessor platform Π. But if TF is to be scheduled on Π' then it is possible to meet all deadlines. It must be that on Π' a processor has a speed which is $1/x$ of the speed of its corresponding processor in Π and $x > \frac{\sqrt{2}}{\sqrt{2}-1}$.

Consider the situation when RM-DU-IS-FF was given TF as input and RM-DU-IS-FF declared FAILURE. There must have been a task $\tau_{failure}$ that was considered when RM-DU-IS-FF declared FAILURE. We can delete all tasks with index greater than $\tau_{failure}$ and we still would have a task set such that the theorem was false. We let τ denote this task set. Clearly we have:

$$Applying\ \tau on\ \Pi\ using\ RM-DU-IS-FF\ declares\ FAILURE \quad (5)$$

and

$$It\ is\ possible\ to\ schedule\ \tau on\ \Pi'\ to\ meet\ deadlines. \quad (6)$$

Let τ_n denote the task that declared failure in (5). Let k denote the number of processors such that $S_p < C_n/T_n$. Due to the sorting performed on line 1 and line 2 we obtain that:

$$For\ every\ (p,i)\ such\ that\ p \in 1,2,\ldots,k\ and\ for\ every\ i \in 1,2,\ldots,n\ it\ holds\ that: S_p < C_i/T_i. \quad (7)$$

From (7) it follows that:

$$When\ RM-DU-IS-FF\ is\ run,\ no\ tasks\ are\ assigned\ to\ processor\ p\ with\ p \in 1,2,..,k. \quad (8)$$

We let U[p] denote the value of the variable U[p] in Algorithm 1 when RM-DU-IS-FF declared FAILURE. Analogously, we let n_p denote the value of the variable $n[p]$ in Algorithm 1 when RM-DU-IS-FF declared FAILURE.

We have that Fact 1 is true.

Fact 1. *When RM-DU-IS-FF declares failure, it holds for $p \in k$+1,k+2,. . .,m: U[p] > ($\sqrt{2}$-1) · S_p.*

Proof. We will prove this using contradiction. Let us assume that the Fact 1 was false. Then it must hold that RM-DU-IS-FF declares failure and $\exists p \in k$+1..m: U[p] $\le$ ($\sqrt{2}$-1) $\times S_p$. Let us consider this processor p and consider the following cases:

Case 1. n_p=0.

Then it follows that U[p]=0 and since $C_n/T_n \le S_p$ it follows that τ_n could have been assigned to processor p. This contradicts (5).

Case 2. n_p=1.

Due to sorting on line 2 in Algorithm 1 we have that $C_n/T_n \le$ ($\sqrt{2}$-1) $\times S_p$. And hence τ_n could have been assigned to processor p. This contradicts (5).

Case 3. $n_p \geq 2$.

We obtain that there is a task τ_j assigned on processor p such that $C_j/T_j \leq \frac{1}{2} \times (\sqrt{2}\text{-}1) \times S_p$. Due to sorting on line 2 in Algorithm 1 we have $C_n/T_n \leq \frac{1}{2} \times (\sqrt{2}\text{-}1) \times S_p$. But then τ_n could have been assigned to processor p, according to Theorem 2. This contradicts (5).

It can be seen that regardless of the case, we obtain a contradiction and it implies that the fact is true. □

From Fact 1 we obtain that when RM-DU-IS-FF declares failure it holds that:

$$\sum_{p=k+1}^{m} (\sqrt{2}-1) \cdot S_p < \sum_{p=k+1}^{m} U[p] \tag{9}$$

Since $\tau_1, \tau_2, \ldots, \tau_{n-1}$ were assigned, we obtain from (9) that:

$$\sum_{p=k+1}^{m} (\sqrt{2}-1) \cdot S_p < \sum_{i=1}^{n-1} \frac{C_i}{T_i} \tag{10}$$

Let us consider two cases.

Case 1. $k \geq 1$.

Let us study a migrative scheduling algorithm that meets all deadlines of τ on Π'. Hence the optimization (1)-(4) has a solution with $l \leq 1$. Fact 2 and Fact 3 reason about this solution.

Fact 2. *For any i, it holds that*

$$\sum_{p=1}^{k} u_{i,p} \leq S'_k$$

Proof. From (2) we obtain that in a migrative schedule where deadlines are met, it holds that:

$$\sum_{p=1}^{m} \frac{u_{i,p}}{S'_p} \leq 1$$

Taking the sum over only a subset yields:

$$\sum_{p=1}^{k} \frac{u_{i,p}}{S'_p} \leq 1$$

Using the fact that the speeds of processors are sorted in ascending order yields:

$$\sum_{p=1}^{k} \frac{u_{i,p}}{S'_k} \leq 1$$

By a simple rewriting this gives us Fact 2. □

Fact 3. *For any i, it holds that*

$$\frac{C_i}{T_i} \leq \frac{x}{x-1} \cdot \sum_{p=k+1}^{m} u_{i,p}$$

Proof. Let i denote the index of any task and let p denote the index of any processor in 1, 2, . . ., k. From (7) we obtain:

$$S_p < \frac{C_i}{T_i} \tag{11}$$

Based on (11),(1) and the sorting of processors, we have:

$$S_k < \sum_{p=1}^{m} u_{i,p} \tag{12}$$

From the assumption on Π and Π' we obtain:

$$S'_k \leq \frac{S_k}{x} \tag{13}$$

Combining Fact 2 and (14) yields:

$$\sum_{p=1}^{k} u_{i,p} \leq \frac{S_k}{x} \tag{14}$$

From (15) we obtain:

$$\sum_{p=1}^{m} u_{i,p} \leq \frac{S_k}{x} + \sum_{p=k+1}^{m} u_{i,p} \tag{15}$$

Combining (13) and (16) yields:

$$\sum_{p=1}^{m} u_{i,p} \leq \frac{\sum_{p=1}^{m} u_{i,p}}{x} + \sum_{p=k+1}^{m} u_{i,p} \tag{16}$$

Rewriting (17) and using (1) yields:

$$\frac{C_i}{T_i} \leq \frac{x}{x-1} \cdot \sum_{p=k+1}^{m} u_{i,p}$$

□

Recall from (10) that when we used partitioning we had:

$$\sum_{p=k+1}^{m} (\sqrt{2}-1) \cdot S_p < \sum_{i=1}^{n-1} \frac{C_i}{T_i}$$

Applying Fact 3 yields:

$$\sum_{p=k+1}^{m} (\sqrt{2}-1) \cdot S_p < \frac{x}{x-1} \cdot \sum_{i=1}^{n-1} \sum_{p=k+1}^{m} u_{i,p}$$

We have $S'_p \leq S_p/x$, where S'_p is the speed of processor p in Π'. Applying this yields:

$$\sum_{p=k+1}^{m} (\sqrt{2}-1) \cdot x \cdot S'_p < \frac{x}{x-1} \cdot \sum_{i=1}^{n-1} \sum_{p=k+1}^{m} u_{i,p}$$

Rewriting (and using the knowledge that x is positive) yields:

$$\sum_{p=k+1}^{m} S'_p < \frac{1}{(\sqrt{2}-1) \cdot (x-1)} \cdot \sum_{i=1}^{n-1} \sum_{p=k+1}^{m} u_{i,p}$$

Since $x > \frac{\sqrt{2}}{\sqrt{2}-1}$ it follows that $\frac{1}{(\sqrt{2}-1)\times(x-1)} < 1$. Using it yields:

$$\sum_{p=k+1}^{m} S'_p < \sum_{i=1}^{n-1} \sum_{p=k+1}^{m} u_{i,p}$$

Swapping the order of the indices of the summation on the right-hand side yields:

$$\sum_{p=k+1}^{m} S'_p < \sum_{p=k+1}^{m} \sum_{i=1}^{n-1} u_{i,p}$$

This requires that there is a $p \in k+1..m$ such that:

$$S'_p < \sum_{i=1}^{n-1} u_{i,p}$$

Dividing by S'_p yields:

$$1 < \sum_{i=1}^{n-1} \frac{u_{i,p}}{S'_p}$$

And hence it is impossible to satisfy (3) and $l \leq 1$. Consequently, a deadline will be missed on Π'. But this contradicts (6). (End of Case 1)

Case 2. $k = 0$.

We have $S'_p \leq S_p/x$, where S'_p is the speed of processor p in Π'. We also have $x > \frac{\sqrt{2}}{\sqrt{2}-1}$. Combining this with (10) yields:

$$\sum_{p=k+1}^{m} (\sqrt{2}-1) \cdot \frac{\sqrt{2}}{\sqrt{2}-1} \cdot S_{p'} < \sum_{i=1}^{n-1} \frac{C_i}{T_i}$$

Simplifying the left-hand side, relaxing it and adding the utilization of τ_n to the right-hand side yields:

$$\sum_{p=k+1}^{m} S_{p'} < \sum_{i=1}^{n} \frac{C_i}{T_i} \qquad (17)$$

From (11) and Lemma 1, it follows that no algorithm can schedule the task set on Π' even if migration is permitted. This contradicts (6). (End of Case 2)

We can see that regardless of the case, we obtain a contradiction and hence Theorem 3 is true. □

Theorem 4. *If we only consider task sets where $\forall i,p$ it holds that: $C_i/T_i \leq S_p$ then $CPT_{RM-DU-IS-FF} \leq 1+\sqrt{2}$.*

Proof. If the theorem was incorrect then it follows (using the same reasoning as in Theorem 3) that there is a task set τ and a computer platform Π and a computer platform Π' such that.

$$\textit{Applying}\ \tau\ \textit{on}\ \Pi\ \textit{using}\ RM-DU-IS-FF\ \textit{declares FAILURE}. \qquad (18)$$

and

$$\textit{It is possible to schedule}\ \tau\ \textit{on}\ \Pi'\ \textit{to meet deadlines}. \qquad (19)$$

and on Π' a processor has a speed which is at most $1/x$ of the speed of its corresponding processor in Π and $x > 1+\sqrt{2}$. Following the reasoning in Theorem 3 we obtain that:

$$\sum_{p=1}^{m} (\sqrt{2}-1) \cdot S_p < \sum_{i=1}^{n-1} \frac{C_i}{T_i} \qquad (20)$$

Combining our knowledge that $S'_p \leq S_p/x$ and $x > 1+\sqrt{2}$ with (20) gives us:

$$\sum_{p=1}^{m} (\sqrt{2}-1) \cdot S'_p \cdot (1+\sqrt{2}) < \sum_{i=1}^{n-1} \frac{C_i}{T_i} \qquad (21)$$

Simplifying yields:

$$\sum_{p=1}^{m} S'_p < \sum_{i=1}^{n-1} \frac{C_i}{T_i} \qquad (22)$$

From (22) and Lemma 1 we obtain that τ misses a deadline on Π'. But this contradicts (19). Hence the theorem is correct. □

5 Experimental Performance Evaluation

The speed competitive ratio offers a guarantee on how much faster processors need to be in order for any task set to be scheduled by RM-DU-IS-FF. This gives us a statement about all task sets. But for individual task sets, let us introduce the following definition.

Definition 1. *Consider a task set τ and a computer platform Π such that τ is feasible (according to (1),(2),(3),(4)) but reducing the speed by an arbitrary amount makes the task set infeasible. Let $\Pi(s)$ denote a computer system where each processor has a speed s times greater than its corresponding processor in Π. We say that this task set has a necessary multiplication of processor speed, s, for algorithm A if the following holds: τ meets deadline when scheduled by A on $\Pi(s)$ but for any $s' < s$, it holds that a deadline is missed when A scheduled τ on $\Pi(s')$.*

We put forward the following hypothesis.

Hypothesis 1. *For many task sets, the necessary multiplication of processor speed for RM-DU-IS-FF is smaller than 3.41 (the value proven in Theorem 3).*

The intuition behind our belief in the truth of the hypothesis is due to the observation in the proof of Theorem 3 that if the task that failed has a low utilization then processors must be "very loaded" and for such task sets the necessary multiplication of processor speed is small.

We will test Hypothesis 1 using simulation experiments on randomly generated task sets. The setup is as follows. n is given by a uniformly distributed random variable in the range $1..MAXn$ and m is given by a uniformly distributed random variable in the range $1..MAXm$. We choose $MAXn$=15 and $MAXm$=15. Tasks are given a utilization which is a uniformly distributed random variable in the range (0,1). The speed of processors are given by a uniformly distributed random variable in the range (0,1). For this task set and this computer platform, we find the value of l that satisfies (1),(2),(3),(4). Then we multiply the speed of every processor by this value of l. Then we obtain a computer platform such that if we would solve (1),(2),(3),(4) then we obtain l=1. This is the task set and the computer platform that we will use.

We apply the algorithm RM-DU-IS-FF on this task set. If it fails then we increase the speed of every processor by 1% and repeat this procedure until we obtain a computer platform where the task set can be scheduled with RM-DU-IS-FF. We have now obtained an approximation of the speed competitive ratio of this task set. We run this procedure for every task set and we do it for 20 000 task sets and obtain the frequency distribution as shown in Figure 1.

We make two observations. First, it can be seen that the speed competitive ratio of every task set in the experiment is less than 1.7. Hence the hypothesis withstood our test. Second, there is a peak, 1.3. This stems from the fact that RM has uniprocessor utilization bound less than 100%. We have run similar experiments with our previously proposed algorithm EDF-DU-IS-FF [2] and observed that for EDF-DU-IS-FF the peak occurs at 1.

6. Previous work

Algorithms in operations research have been proposed for scheduling jobs with no real-time requirements assuming that all jobs arrive at the same time and the goal is to minimize the time when all jobs have been finished. (See for example [11].) A solution to this problem can be used for scheduling periodically arriving tasks with deadlines [3]. But unfortunately, that algorithm [3] allows task migration and hence it cannot solve our problem.

The problem of partitioning a task set on a uniform multiprocessor has been considered previously when using EDF on each processor [9] or using RM on each processor [8]. This is a similar problem as we addressed in this paper. We find a drawback with those algorithms and analysis though. The algorithms are analyzed by extending the utilization bound from identical multiprocessors. But their utilization bound is not a single number; it is a function of the maximum C_i/T_i of tasks. This causes a large amount of pessimism when (i) the difference in speeds of processors is very large and (ii) the maximum C_i/T_i is large. This pessimism is a consequence of neither the algorithm, nor the analysis techniques, but it is a consequence of the definition of the utilization bound in uniform multiprocessors. Example 2 illustrates why the utilization bound is unsuitable as a performance metric.

Example 2. *Consider two tasks to be scheduled on $m \geq 4$ processors. The task set is characterized by T_1 = 1, C_1 = 1, T_2 = 1, C_2 = $1/L+1/L^2$ and the processors have the speed S_1 = 1 and $S_2 = S_3 = \ldots = S_m = 1/L$, where $L \geq 2$. In order to meet deadlines, it is necessary that τ_1 is assigned to processor 1. Now, it can be seen that wherever τ_2 is assigned, the utilization of that processor becomes more than its speed and hence it is impossible to meet deadlines. By letting L=$\sqrt{m}$ and m approach infinity we obtain that $\frac{\sum_{i=1}^{n} C_i/T_i}{\sum_{p=1}^{m} S_p} \rightarrow 0$. Hence, if the utilization bound is a single number (that is not a function of the maximum utilization of tasks) then every partitioned algorithm for uniform multiprocessors have a utilization bound of 0. Consequently, such a utilization bound cannot be used to distinguish between "'good"' and "'bad"' algorithms for assigning tasks to processors. It would be possible to use the utilization bound as a performance metric if the maximum C_i/T_i could be fixed and only such task sets are considered. But it is unclear how such a restriction should be chosen and how*

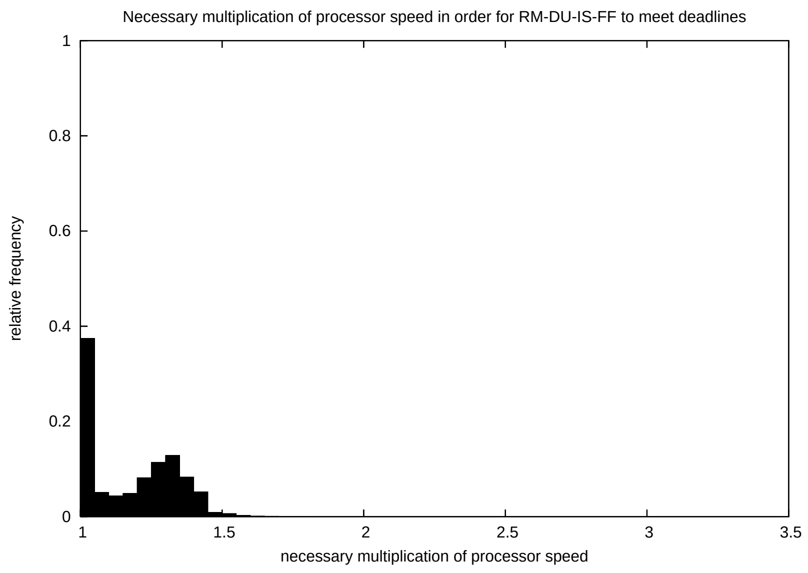


Figure 3. The necessary multiplication of processor speed.

to justify which threshold of maximum C_i/T_i that should be used.

The problem we address can be solved using task assignment algorithms for heterogeneous multiprocessors [6, 5]. The algorithm in [6] uses exhaustive enumeration of ”heavy tasks” and this leads to a time complexity of O(m^m). The other algorithm [5] has polynomial time-complexity but it is high; it requires that a linear program is solved. Real-time scheduling algorithms for uniform multiprocessors have been proposed [10, 7] but unfortunately they require that tasks can migrate.

We have studied uniform multiprocessors and we studied how much extra processing power must be given to our algorithm to ensure that our algorithm meets deadlines for every task set which an optimal algorithm can schedule to meet deadlines. This type of analysis was originally proposed in [12, 15] but for a uniprocessor [12] and a multiprocessor where all processors have the same speed [15]. Such studies used the notion of ”‘resource augmentation factor”’; this notion was used to study online scheduling, that is, scheduling where the characteristics of arriving jobs are unknown before they arrive. Our problem addresses offline scheduling, that is, all characteristics of jobs (except the exact arrival time) are known. We characterized the performance using the notion of the ”‘speed competitive ratio”’. The notion of ”‘speed competitive ratio”’ and ”‘resource augmentation factor”’ are very similar; both prove that giving more resources can make the studied scheduling algorithm A to succeed if there is any other scheduling algorithm that succeeds. We used notion ”‘speed competitive ratio”’ to emphasize that our problem is an offline problem rather than an online problem and that our problem addressed is more similar to bin-packing problems. (The notion of competitive ratio in bin-packing gives ”‘extra bins”’ and this notion is unclear in uniform multiprocessor scheduling because it is unclear what the size of these ”‘extra bins”’ should be.)

7. Conclusions

We have presented an algorithm to schedule sporadically arriving tasks on a uniform multiprocessor and we have proven its speed competitive ratio. It is at most $\frac{\sqrt{2}}{\sqrt{2}-1} \approx 3.41$. This is the first proven speed competitive ratio in real-time scheduling on uniform multiprocessors where RM is used on each processor and migration is not allowed.

We left open the questions (i) whether the bounds on speed competitive ratio of RM-DU-IS-FF that is proven in this paper is tight and (ii) whether it is possible to design a better partitioning scheme.

Acknowledgements

We are grateful to the reviewers for suggested improvements of the paper. This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para Ciência e Tecnologia - FCT) and the ARTIST2 Network of Excellence on Embedded Systems Design.

References

[1] B. Andersson, S. K. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *IEEE Real-Time Systems Symposium*, 2001.

[2] B. Andersson and E. Tovar. Competitive analysis of partitioned scheduling on uniform multiprocessors. In *IEEE International Workshop on Parallel and Distributed Real-Time Systems*, 2007.

[3] S. K. Baruah. Scheduling periodic tasks on uniform multiprocessors. In *12th Euromicro Conference on Real-Time Systems*, 2000.

[4] S. K. Baruah. Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In *25th IEEE International Real-Time Systems Symposium*, 2004.

[5] S. K. Baruah. Partitioning real-time tasks among heterogeneous multiprocessors. In *International Conference on Parallel Processing*, 2004.

[6] S. K. Baruah. Task partitioning upon heterogeneous multiprocessor platforms. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004.

[7] S. K. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Trans. Computers*, 52:966–970, 2003.

[8] V. Darera and L. Jenkins. Utilization bounds for RM scheduling on uniform multiprocessors. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006.

[9] S. Funk and S. K. Baruah. Task assignment on uniform heterogeneous multiprocessors. In *17th Euromicro Conference on Real-Time Systems*, 2005.

[10] S. Funk, J. Goossens, and S. K. Baruah. On-line scheduling on uniform multiprocessors. In *IEEE Real-Time Systems Symposium*, 2001.

[11] T. Gonzalez and S. Sahni. Preemptive scheduling of uniform processor systems. *Journal of the ACM*, 25, 1978.

[12] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *IEEE Symposium on Foundations of Computer Science*, 1995.

[13] C. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20:46–61, 1973.

[14] D.-I. Oh and T. Baker. Utilization bounds for n-processor rate monotone scheduling with stable processor assignment. *Real Time Systems*, 15(1), 1998.

[15] C. A. Phillips, C. Stein, E. Tornh, and J. Wein. Optimal time-critical scheduling via resource augmentation. *ACM Symposium on Theory of Computing*, 1997.