IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

Competitive Analysis of Partitioned Scheduling on Uniform Multiprocessors

Björn Andersson

Eduardo Tovar

TR-061101

Version: 1.0

Date: Nov 2006

Competitive Analysis of 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}@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. 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 three times faster.

Competitive Analysis of Partitioned Scheduling on Uniform Multiprocessors

Björn Andersson and Eduardo Tovar
IPP Hurray Research Group
Polytechnic Institute of Porto, Portugal
{bandersson,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. 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 three times faster.

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 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 $0 \leq C_i$ and $0 \leq T_i$, and that T_i and C_i are real numbers. Note that we permit $T_i \leq C_i$.

The scheduling algorithm is allowed to preempt the execution of a task and there is no cost associated with preemption. Task 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 can execute at most one job at a time, 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 [1]. 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* [2]. 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 [2] and on multiprocessors with identical processors [3, 4] cannot be applied on uniform processors. For this reason, we will instead use another commonly-used [5, 6] performance metric: the *competitive factor*.

The competitive factor of an algorithm A is denoted as CPT_A. It is a number such that for every task set τ and for every uniform multiprocessor system Π, characterized by its speed S_p, it holds that if it is possible to design an algorithm that meets all deadlines of τ on Π' then A meets all deadlines of τ on Π, where Π is a uniform multiprocessor system where each processor has a speed CPT_A greater than the corresponding processor in Π'.

A low competitive factor indicates high performance. A competitive factor of 1 is the best achievable. And a competitive factor of 2 is (as we will see) the best achievable for scheduling algorithms that do not allow migration. If a scheduling algorithm has a finite competitive factor then one can solve every problem instance using processors that are sufficiently fast. If no finite competitive factor has been proven for a scheduling problem then one cannot know if faster processors will ever help. Unfortunately, with the current state of art in partitioned scheduling on uniform multiprocessor, there is no algorithm with a proven finite competitive factor.

Therefore, in this paper we propose a partitioned scheduling algorithm for uniform multiprocessors; it allows preemption and it uses Earliest-Deadline-First (EDF) [2] on each processor. We prove its competitive factor: it is at most three.

The remainder of this paper is organized as follows. Section 2 discusses design issues for uniform multiprocessors. Section 3 discusses the problem of deciding whether it is possible to schedule a task set on a uniform multiprocessor assuming that the scheduling algorithm is permitted to migrate tasks. Section 4 presents our new algorithm which does not migrate tasks. We also prove its competitive factor. This proof uses results in Section 3 on scheduling where migration is permitted. Section 5 discusses the ability of previous work to solve the addressed problem. Section 6 gives conclusions.

2. Design issues

Recall that task 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. 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. The run-time scheduling is trivial. It is well-known that preemptive Earliest-Deadline-First (EDF) is optimal on a uniprocessor with our task model; that is, it meets deadlines if there is any uniprocessor scheduling algorithm that meets deadlines. For this reason, we will, in the remainder of the paper, assume that preemptive EDF is used on each processor. For convenience we will refer to EDF with the meaning of preemptive EDF.

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. This is illustrated in Example 1.

Example 1. Consider $n = 3$ tasks to be scheduled on $m = 2$ processors. The task set is characterized by $T_1 = 1$, $C_1 = 0.34$, $T_2 = 1$, $C_2 = 0.34$, $T_3 = 1$, $C_3 = 0.34$; and the processors have the speed $S_1 = S_2 = 1$. If all tasks are assigned to processor 1 then the utilization on that processor is 1.02. This causes a deadline miss. Consequently, an algorithm for assigning tasks to processors must use a schedulability test according to the uniprocessor scheduling algorithm to be used at run-time. □

We saw in Example 1 that the task assignment algorithm must use a schedulability test when making decisions. For EDF it is known [2] that:

Theorem 1. Let p be a processor of speed $S_p = 1$. If $\sum C_i/T_i \leq 1$ and tasks are scheduled with EDF on p then all deadlines are met. □

We can easily remove the restriction $S_p = 1$ from Theorem 1.

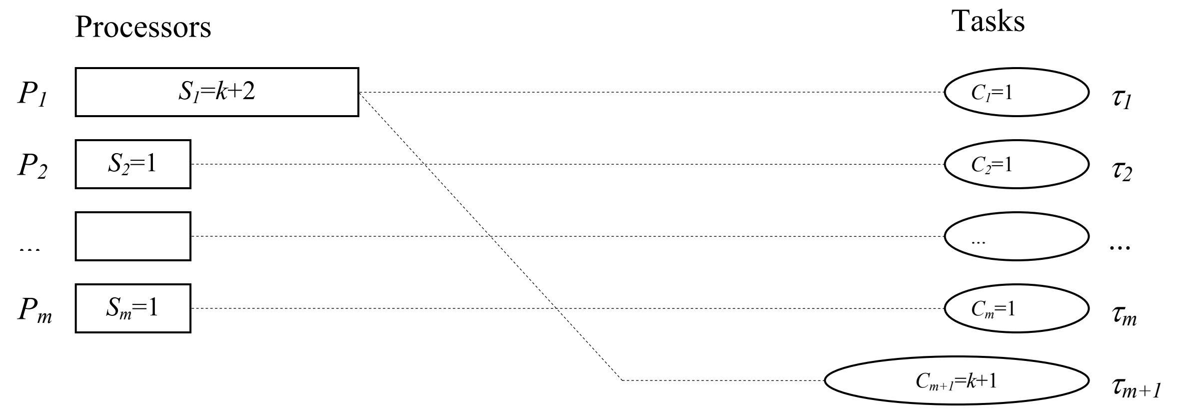
Theorem 2. Let p be a processor of speed S_p. If $\sum C_i/T_i \leq S_p$ and tasks are scheduled with EDF on p then all deadlines are met. □

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 competitive factor. Example 2 illustrates this.

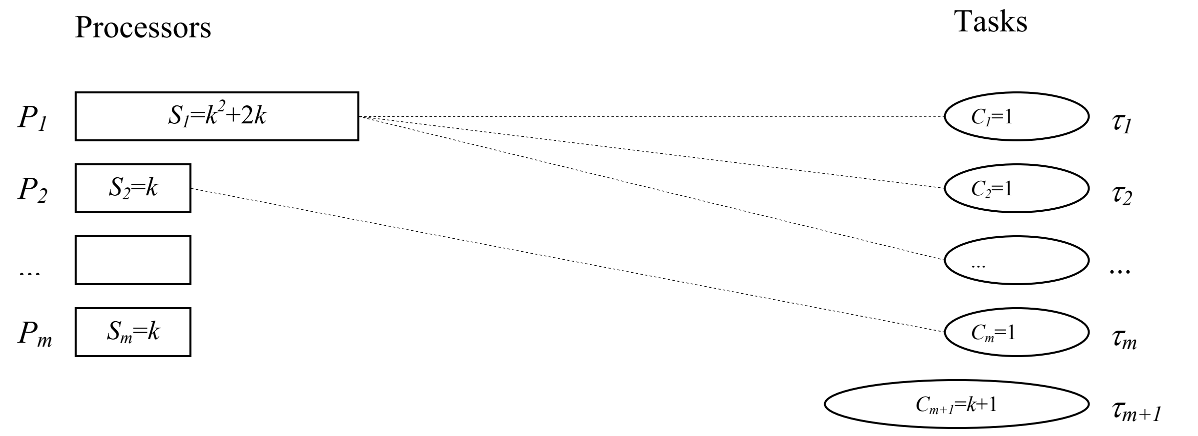
Example 2. Let k be an arbitrary integer such that $k \geq 2$. Consider $n = k^3+1$ tasks to be scheduled on $m = k^3$ processors. All tasks have $T_i = 1$, $\forall i \in 1..m+1$. Tasks with $\forall i \in 1..m$, have $C_i = 1$ and the task τ_{m+1} has $C_{m+1} = k+1$. Processor 1 has the speed $S_1 = k+2$ 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 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 $S_1 = k^2+2k$ and processors $2,3,4,\ldots,m$ will have speed $S_p = k$. The speed of processor 1 is not enough to host all the tasks $\tau_1,\tau_2,\tau_3,\ldots,\tau_m$ because their cumulative utilization is k^3 and this exceeds the speed of processor 1, which is $S_1 = k^2+2k$ (it is true that $k^3 \geq k^2+2k$ since $k \geq 2$). 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 $2,3,\ldots,m$. But τ_{m+1} cannot be assigned to a processor with index $2,3,\ldots,m$ 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 $k \geq 2$. By letting $k \to \infty$ we obtain that the competitive factor 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



(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.

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

assignment algorithm makes decisions. □

We saw in Example 2 that it is important to take the speed of processors into account when assigning tasks to processors. In particular, if a task can be assigned to a processor such that this task occupies a large fraction of the processing capacity of that processor, then it is beneficial to assign the task to that processor. Considering that we will use the competitive factor as a performance metric, it is interesting to find out how good performance can be achieved. Clearly, we want as low competitive factor as possible, and clearly a competitive factor less than 1 is impossible. But since we study scheduling with no migration, a competitive factor of two is the best achievable, as it will be shown in Example 3.

Example 3. Observe Figure 2. Consider $n = m+1$ tasks to be scheduled on m processors. All tasks have $T_i = 1$, $C_i = m/(m+1)$ $\forall i \in 1..m+1$. All processors have speed $S_p = 1$. It can be seen that these tasks can be scheduled to meet deadlines with an algorithm that allows task migration because $\sum C_i/T_i \leq m$ and all processors are identical. Figure 2a shows this. Let us now try to schedule these tasks without migration on processors of speed $S_p = 2m/(m+1)-\varepsilon$, where $\varepsilon > 0$. It is necessary that two or more tasks are assigned to the same processor. On that processor, the utilization exceeds the speed of the processor and hence a deadline is missed. We can do this reasoning for any $m \geq 1$

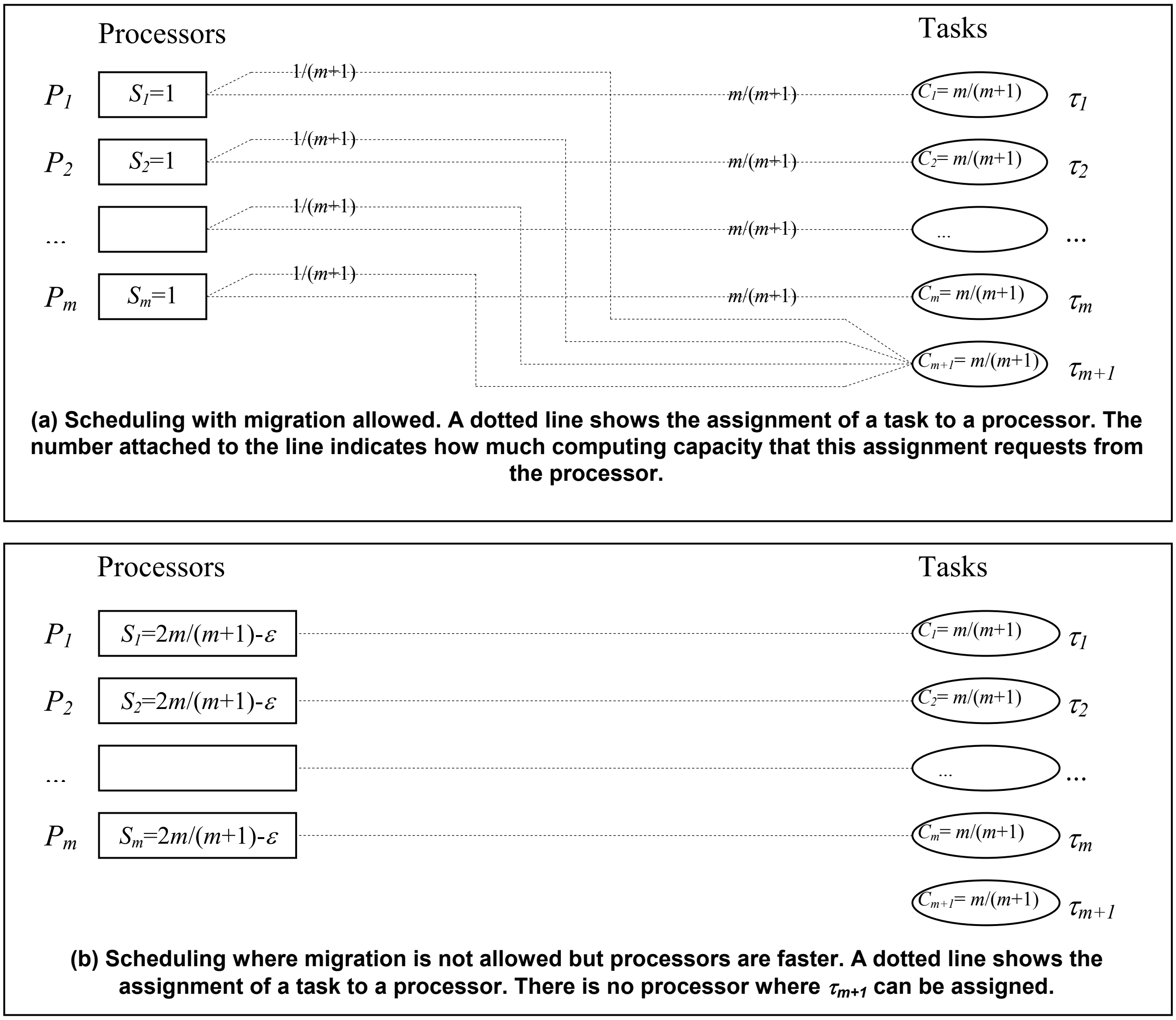


Fig. 2. The competitive factor of every partitioned scheduling algorithm is at least 2.

and for any $\varepsilon > 0$. Letting $m \to \infty$ and $\varepsilon \to 0$ yields that a deadline is missed although the speed is arbitrarily close to two. Hence, it is impossible to achieve a competitive factor less than 2 for partitioned scheduling. □

3. Optimal Scheduling With Migration

We will now discuss feasibility testing of scheduling with migration; 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. We will state those conditions for a heterogeneous multiprocessor platform (in Section 3.1) and then we will state them (in Section 3.2) for uniform platforms. The latter is useful for proving the competitive factor of the new algorithm in Section 4.

3.1. Heterogeneous Multiprocessor Platforms

The problem of feasibility testing on a heterogeneous multiprocessor platform has been studied previously [7]. We define $r_{i,p}$ as follows: on a heterogeneous multiprocessor platform, a task τ_i executing on processor p for L time units, performs $r_{i,p} \times L$ units of work. Let $x_{i,p}$ denote the fraction of time that task τ_i spends on processor p. It holds that a task set is feasible on a heterogeneous multiprocessor platform if and only if $l \leq 1$ for the following optimization problem.

```
minimize l
subject to:
```

$$\forall i \in \{1,2,...,n\}: \sum_{p=1}^{m} x_{i,p} \times r_{i,p} = \frac{C_i}{T_i}$$

and

$$\forall i \in \{1,2,...,n\}: \sum_{p=1}^{m} x_{i,p} \leq l$$

and

$$\forall p \in \{1,2,...,m\}: \sum_{i=1}^{n} x_{i,p} \leq l$$

3.2. Uniform Multiprocessor Platforms

We can specialize the feasibility analysis in Section 3.1 to uniform multiprocessors. We have $\forall p$: $r_{1,p} = r_{2,p} = r_{3,p} = \ldots = r_{n,p} = S_p$, where S_p is the speed of processor p and $r_{i,p}$ is the parameter from Section 3.1. The feasibility test can then be formulated as follows: 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:

$$\forall i \in \{1,2,...,n\}: \sum_{p=1}^{m} x_{i,p} \times S_p = \frac{C_i}{T_i}$$

and

$$\forall i \in \{1,2,...,n\}: \sum_{p=1}^{m} x_{i,p} \leq l$$

and

$$\forall p \in \{1,2,...,m\}: \sum_{i=1}^{n} x_{i,p} \leq l$$

Let us substitute $x_{i,p} \times S_p$ with $u_{i,p}$. Then, the feasibility test can then be reformulated as follows: 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:

$$\forall i \in \{1,2,..., n\}: \sum_{p=1}^{m} u_{i,p} = \frac{C_i}{T_i} \quad (1)$$

and

$$\forall i \in \{1,2,...,n\}: \sum_{p=1}^{m} \frac{u_{i,p}}{S_p} \leq l \quad (2)$$

and

$$\forall p \in \{1,2,...,m\}: \sum_{i=1}^{n} \frac{u_{i,p}}{S_p} \leq l \quad (3)$$

From (1), (2) and (3) 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: We know from the assumption of the lemma that there is a task set τ and a uniform multiprocessor Π such that:

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

Applying (1) yields:

$$\sum_{p=1}^{m} S_p < \sum_{i=1}^{n} \sum_{p=1}^{m} u_{i,p}$$

and swapping the summation order yields:

$$\sum_{p=1}^{m} S_p < \sum_{p=1}^{m} \sum_{i=1}^{n} u_{i,p}$$

This requires that there is a p such that:

$$S_p < \sum_{i=1}^{n} u_{i,p}$$

Dividing by S_p yields:

$$1 < \sum_{i=1}^{n} \frac{u_{i,p}}{S_p}$$

Hence it is impossible to satisfy (3) and $l \leq 1$.

Consequently, a deadline will be missed. This proves the lemma. □

4. The New Algorithm

The new algorithm is described in Figure 3. It is called EDF-DU-IS-FF because it uses EDF 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 11 is the schedulability test from Theorem 2. It is straightforward to see that the algorithm has the time complexity O($n \times m + n \times \log n$). The performance of EDF-DU-IS-FF is given by Theorem 3.

Theorem 3. $CPT_{EDF\text{-}DU\text{-}IS\text{-}FF} \leq 3$

Proof: We can prove it using contradiction. We will do so and show that a failed task set must request more than 50% 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 proves the theorem. Let us elaborate this reasoning.

```
1.   sort processors such that S1≤S2≤…≤Sm
2.   sort tasks such that C1/T1≥C2/T2≥…≥Cn/Tn
3.   for all p in 1..m do
4.     U[p] := 0
5.   end for
6.   i := 1
7.   while (i<=n) do
8.     p := 1
9.     allocated := FALSE
10.    while (p<=m) and (allocated=FALSE) do
11.      if U[p]+ Ci/Ti<=Sp then
12.        assign task i to processor p
13.        U[p] := U[p]+ Ci/Ti
14.        allocated := TRUE
15.        i := i + 1
16.      else
17.        p := p + 1
18.      end if
19.    end while
20.    if (allocated=FALSE) then
21.      declare FAILURE
22.    end if
23.  end while
24.  declare SUCCESS
```

Fig. 3. EDF-DU-IS-FF, a task assignment algorithm for a uniform multiprocessor.

If the theorem was false then there exists a task set TF such that EDF-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>3.

Consider the situation when EDF-DU-IS-FF was given TF as input and EDF-DU-IS-FF declared FAILURE. There must have been a task $\tau_{failure}$ that was considered when EDF-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 τ on Π using EDF-DU-IS-FF declares FAILURE (4)

and

It is possible to schedule τ on Π′ to meet deadlines (5)

It was task τ_n that declared failure in (4). 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 $p\in\{1,2,..,k\}$ and for $i\in\{1,2,..,n\}$ it holds that: $S_p < C_i/T_i$. (6)

From (6) it follows that:

When EDF-DU-IS-FF is run, no tasks are assigned to processor p with $p\in\{1,2,..,k\}$. (7)

Let us now consider τ_n, the task that caused failure for EDF-DU-IS-FF. We know that:

For $p\in\{k+1,2,..,m\}$, it holds that $U[p]+C_n/T_n > S_p$ (8)

Observe from (8) that τ_n could not be assigned to any of the processors k+1,k+2,…,m, despite the fact that $C_n/T_n \le S_p$ for those processors. Hence we have that:

When EDF-DU-IS-FF declares FAILURE, for each processor $p\in\{k+1,k+2,..,m\}$ it holds that: there is at least one task assigned to processor p. (9)

We have that Fact 1 is true.

Fact 1. When EDF-DU-IS-FF declares failure, it holds that $\forall p \in \{k+1,k+2,\ldots,m\}$: $U[p]>0.50\times S_p$.

Proof: If Fact 1 was false then there must exist a processor p with $U[p] \le 0.50\times S_p$. We know from (9) that there is at least one task assigned to processor p. Hence there is a task with $C_i/T_i \le 0.50\times S_p$ assigned to processor p. Due to the sorting of tasks we have that $C_n/T_n \le C_i/T_i$ and it leads to $C_n/T_n \le 0.50\times S_p$. But then it would be possible for τ_n to be assigned to processor p and we know that it cannot happen since EDF-DU-IS-FF declared failure. This is a contradiction and it proves the fact. (End of proof of Fact 1) □

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

$$\sum_{p=k+1}^{m} 0.5\times S_p < \sum_{p=k+1}^{m} U[p] \tag{10}$$

Since $\tau_1,\tau_2,\ldots,\tau_{n-1}$ were assigned, we obtain from (10) that:

$$\sum_{p=k+1}^{m} 0.5\times S_p < \sum_{i=1}^{n-1}\frac{C_i}{T_i} \tag{11}$$

Let us consider two cases

<u>**Case 1.** k = 0.</u>

We have $S_p' \le S_p/x$, where S_p' is the speed of processor p in Π′. We also have $x > 3$. Combining this with (11) yields:

$$\sum_{p=1}^{m} 0.5\times 3\times S_p' < \sum_{i=1}^{n-1}\frac{C_i}{T_i}$$

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

$$\sum_{p=1}^{m} S_p' < \sum_{i=1}^{n}\frac{C_i}{T_i} \tag{12}$$

From (12) and Lemma 1, it follows that no algorithm can schedule the task set on Π′ even if migration is permitted. This contradicts (5). (End of Case 1)

<u>**Case 2.** $k \ge 1$</u>

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

Fact 2. For any i, it holds that

$$\sum_{p=1}^{k} u_{i,p} \le S_k'$$

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

$$\sum_{p=1}^{m} \frac{u_{i,p}}{S_p'} \le 1$$

Taking the sum over only a subset yields:

$$\sum_{p=1}^{k} \frac{u_{i,p}}{S_p'} \le 1$$

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

$$\sum_{p=1}^{k} \frac{u_{i,p}}{S_k'} \le 1$$

By a simple rewriting this gives us Fact 2.
(End of proof of Fact 2.) □

Fact 3. For any i, it holds that

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

Proof: From (6) we obtain:

$$S_k < \frac{C_i}{T_i} \qquad (13)$$

Based on (13) and (1) we have:

$$S_k < \sum_{p=1}^{m} u_{i,p} \qquad (14)$$

From the assumption on Π and Π′ we obtain:

$$S_k' < \frac{S_k}{x} \qquad (15)$$

Combining Fact 2 and (15) yields:

$$\sum_{p=1}^{k} u_{i,p} \le \frac{S_k}{x} \qquad (16)$$

From (16) we obtain:

$$\sum_{p=1}^{m} u_{i,p} \le \frac{S_k}{x} + \sum_{p=k+1}^{m} u_{i,p} \qquad (17)$$

Combining (14) and (17) yields:

$$\sum_{p=1}^{m} u_{i,p} \le \frac{\sum_{p=1}^{m} u_{i,p}}{x} + \sum_{p=k+1}^{m} u_{i,p} \qquad (18)$$

Rewriting (18) and using (1) yields:

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

(End of proof of Fact 3.) □

Recall from (11) that when we used partitioning we had:

$$\sum_{p=k+1}^{m} 0.5 \times S_p < \sum_{i=1}^{n-1} \frac{C_i}{T_i}$$

Applying Fact 3 yields:

$$\sum_{p=k+1}^{m} 0.5 \times S_p < \frac{x}{x-1} \times \sum_{i=1}^{n-1} \sum_{p=k+1}^{m} u_{i,p}$$

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

$$\sum_{p=k+1}^{m} 0.5 \times x \times S_p' < \frac{x}{x-1} \times \sum_{i=1}^{n-1} \sum_{p=k+1}^{m} u_{i,p}$$

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

$$\sum_{p=k+1}^{m} S_p' < \frac{2}{x-1} \times \sum_{i=1}^{n-1} \sum_{p=k+1}^{m} u_{i,p}$$

Since $x > 3$ we obtain that $2/(x-1)<1$. 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 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 $l \le 1$.
Consequently, a deadline will be missed on Π'. But this contradicts (5). (End of Case 2)

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

5. 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 [8].) A solution to this problem can be used for scheduling periodically arriving tasks with deadlines [1]. But unfortunately, that algorithm [1] allows task migration and hence it cannot solve our problem.

The problem of partitioning a set of tasks on a uniform multiprocessor has been considered previously [9, 10]. This is the same problem as we addressed in this paper. We find two drawbacks with those algorithms and analysis though. First, 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 neither a consequence of the algorithm, nor the analysis techniques but it is a consequence of the definition of the utilization bound in uniform multiprocessors. The second drawback of the above mentioned previous work [9, 10] is that their competitive factor is infinite. The algorithms use First-Fit or Any-Fit; this is a good design. However, the algorithms sort processors in increasing order of speed; this makes it possible for the behavior of Example 2 to occur and it causes the competitive factor to be infinite.

The problem we address can be solved using task assignment algorithms for heterogeneous multiprocessors [11, 12]. The algorithm in [11] uses exhaustive enumeration of "heavy tasks" and this leads to a time complexity of $O(m^m)$. The other algorithm [12] has polynomial time-complexity but it is high; it requires that a linear program is solved. Neither of them proves a competitive factor.

A competitive factor has already been proven for scheduling real-time tasks on uniform multiprocessors [5, 6]; one of the algorithms has a competitive factor of two [5]. In addition it has the advantage of being proven not just for the sporadic task model but for the more generic model of aperiodic jobs where the scheduling algorithm has no knowledge of jobs arriving in the future. Unfortunately, it requires that tasks can migrate.

6. Conclusions

We have presented an algorithm to schedule sporadically arriving tasks on a uniform multiprocessor and we have proven its competitive factor. It is at most three. This is a significant result because it is the first proven competitive factor in real-time scheduling on uniform multiprocessors where migration is not allowed.

We left open the questions (i) Is it possible to achieve a competitive factor of two without migration? (ii) What is the competitive factor when rate-monotonic [2] is used on each processor?

References

[1] S. K. Baruah, "Scheduling periodic tasks on uniform multiprocessors," in *12th Euromicro Conference on Real-Time Systems*. Stockholm, Sweden, 2000.

[2] C.L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the Association for Computing Machinery* vol. 20, pp. 46-61, 1973.

[3] S.K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations Research*, vol. 26, pp. 127-140, 1978.

[4] B. Andersson, S. K. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proc. of the IEEE Real-Time Systems Symposium*. London, 2001.

[5] S. Funk, J. Goossens, and S. K. Baruah, "On-Line Scheduling on Uniform Multiprocessors," in *IEEE Real-Time Systems Symposium*. London, UK, 2001.

[6] S. K. Baruah and J. Goossens, "Rate-Monotonic Scheduling on Uniform Multiprocessors," *IEEE Trans. Computers*, vol. 52, pp. 966-970, 2003.

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

[8] T. Gonzalez and S. Sahni, "Preemptive scheduling of uniform processor systems," *Journal of the ACM*, vol. 25, pp. 92, 1978.

[9] S. Funk and S. K. Baruah, "Task assignment on uniform heterogeneous multiprocessors," in *17th Euromicro Conference on Real-Time Systems, 2005.* Palma de Mallorca, Balearic Islands, Spain, 2005.

[10] V. Darera and L. Jenkins, "Utilization Bounds for RM Scheduling on Uniform Multiprocessors," in *The 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. Sydney, Australia, 2006.

[11] S. K. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *10th IEEE Real-Time and Embedded Technology and Applications Symposium*. Toronto, Canada, 2004.

[12] S. K. Baruah, "Partitioning real-time tasks among heterogeneous multiprocessors," in *International Conference on Parallel Processing*. Montréal, Québec, Canada, 2004.