Sustainability in static-priority restricted-migration scheduling

Frédéric Fauberteau, Serge Midonnet

▶ To cite this version:

Frédéric Fauberteau, Serge Midonnet. Sustainability in static-priority restricted-migration scheduling. RACS 2012, Oct 2012, San Antonio, United States. pp.416-421, 2012, <10.1145/2401603.2401692>. <hal-00797909>

HAL Id: hal-00797909

https://hal.archives-ouvertes.fr/hal-00797909

Submitted on 7 Mar 2013

Sustainability in static-priority restricted-migration scheduling

Frédéric Fauberteau
CEA, List,
Embedded Real-Time Systems Laboratory
Point Courrier 94
Gif-sur-Yvette, F-91191 France
frederic.fauberteau@cea.fr

Serge Midonnet
Université Paris-Est
LIGM, UMR CNRS 8049
5, bd Descartes – Champs-sur-Marne
77454 Marne-la-Vallée CEDEX 2, France
serge.midonnet@univ-paris-est.fr

ABSTRACT

In this paper, we focus on the static-priority scheduling of periodic hard real-time tasks upon identical multiprocessor platforms. In order to bound the inter-processor migrations, we consider the restricted-migration scheduling policy for which a task is allowed to migrate only at job boundaries. Several jobs of the same task can then be assigned on different processors but a given job can not migrate. It has been shown that this scheduling policy can suffer from scheduling anomalies. These anomalies occur when a decrease in execution requirement of a job causes a deadline miss. We present a static-priority restricted-migration scheduling algorithm and we prove it does not suffer from these anomalies. We also review the scheduling anomalies according to the scheduling tests for this algorithm.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; J.7 [**Computers in other systems**]: Real time

General Terms

Algorithms

Keywords

Real-Time, Scheduling, Multiprocessor, Sustainability

1. INTRODUCTION

The recent improvements in the embedded systems induce more powerful and complex systems on which we can reasonably run a modern operating system. In the case of the real-time embedded applications, a Real-Time Operating System (RTOS) can be a good solution to provide security and reliability. A RTOS also enables to implement an advanced scheduler without suffering from intolerable overrides due to costly computations of scheduling decisions. In the case of the multiprocessor embedded systems, the choice of the scheduler is very important in order to reduce the number of context switches and migrations.

The literature about real-time multiprocessor scheduling often states two main approaches (a third one is presented bellow) to schedule sets of tasks. These two multiprocessor scheduling approaches are referenced as *global* and *partitioned* scheduling. The first one allows inter-processor migrations while the second one prevents them. Although the superiority of partitioned scheduling over global scheduling has been highlighted according to the best known schedulability tests [2], these two approaches are incomparable. Some set of tasks can be scheduled with partitioned algorithm and not by a global one and vice versa. It remains difficult to choose among one of these two approaches. As presented in [7], the intermediate *restricted-migration* approach has to be considered. This approach allows task-level migrations but no job-level migrations. A non-hybrid [1] multiprocessor scheduling algorithm can be classified among one of the following classes:

- **partitioned**: Each task is assigned on a processor. For a given task, each job is released on the processor on which the task has been assigned. No migration is allowed (*e.g.* [19], *EDF-FF* [17]).
- **restricted-migration**: At each time a job is released, it can be assigned on a different processor. The task-level migrations are allowed but a job can not migrate during its execution (*e.g.* *r-EDF* [5], *r-SP_wl* [8]).
- **global**: The job-level migrations are allowed. A job can start on a first processor and finishes on a second one (*e.g.* *RM-US*$[m/(3m-2)]$ [1]), *PriD* [12]).

This classification reminds the one made for the three main classes of uniprocessor scheduling algorithm. A uniprocessor scheduling algorithm can be classified among one of the following classes:

- **fixed-task-priority** or **static-priority**: A priority is attributed to each task composing the system. For a given task, each job inherits the priority of the task which generates it. The priority of a job is constant during its execution. (*e.g.* *Rate-Monotonic* [16], *Deadline-Monotonic* [15], ...).

[1]An hybrid algorithm exploits behavior from two different classes for two different subsets of tasks (See [11] for more details)

- **fixed-job-priority**: At each time a job is released, its priority is computed and it is constant during the execution of the job (*e.g. Earliest-Deadline-First* [16]).
- **dynamic-priority**: The priority of any job can change during its execution (*e.g. Least-Laxity-First* [18])

We refer to a **fixed-priority** scheduling algorithm in the case where the scheduling algorithm is either fixed-task-priority or fixed-job-priority and only jobs are considered.

In multiprocessor context, a non-hybrid scheduling algorithm can reasonably be referred according to both how the prioritization is done (fixed-task / fixed-job / dynamic) and how the migrations are allowed (no / task-level / job-level).

Although the restricted-migration scheduling has been less studied than the two main multiprocessor scheduling approaches, we can exhibit some interesting works. Ha and Liu have studied the fixed-priority restricted-migration scheduling of a set of jobs and have shown this kind of scheduling is prone to a problem of predictability (scheduling anomaly which occurs when the execution requirement of jobs is decreased). Baruah and Carpenter have proposed *r-EDF*, an EDF-based algorithm for periodic tasks on identical processors [5]. In the same paper, they also proposed an extension of *PriD* [12] to *r-PriD* (which exploits restricted-migration). Funk has studied in her thesis [10] the restricted-migration scheduling approach in the case of an uniform multiprocessor. Fisher has studied in his thesis [9] the restricted-migration scheduling approach in the case of scheduling of a set of jobs. Fauberteau *et al.* have proposed *r-SP_wl*, a static-priority algorithm for periodic tasks on identical processors [8]. The contribution of this paper is to propose a proof of sustainability for *r-SP_wl* algorithm in the case of the constrained deadline period sets of tasks.

The remainder of this paper is organized as follows. In Section 2, we remind the model that we consider and we introduce the notations that we use. In Section 3, we define the concept of sustainability and we give an example illustrating a scheduling anomaly which can occur with a non-predictable algorithm. In Section 4, we present our static-priority restricted-migration scheduling algorithm and we prove that it is sustainable according to our considered model. We conclude in Section 5 by giving some future work perspectives.

2. SYSTEM MODEL

We consider a set τ of n constrained deadline periodic tasks indexed from 1 to n. (*i.e.* $\tau = \{\tau_1, \ldots, \tau_n\}$). The index of the task is in inverse proportion to its priority. The less is the task index the higher its priority is.

A task τ_i is characterized by:

- its Worst Case Execution Time (WCET) denoted C_i,
- its relative deadline denoted D_i,
- its period denoted T_i.

A task τ_i (C_i, D_i, T_i) consists of an infinite recurrence of jobs where each job $J_{i,k}$ is characterized by:

- its release time denoted $r_{i,k}$,
- its execution requirement denoted $e_{i,k}$ such that $0 \leq e_{i,k} \leq C_i$,
- its absolute deadline denoted $d_{i,k}$ such that $d_{i,k} = r_{i,k} + D_i$.

For readability, we can consider a job independently of the task which generates it and we denote it $J_k(r_k, e_k, d_k)$.

By definition, the utilization of a periodic task is given by $u_i = \frac{C_i}{T_i}$. By extension, we define the utilization of a set of tasks as the sum of the utilization of the tasks which compose it and its value is given by $U(\tau) = \sum_i^n u_i$. We also denote $U_{max}(\tau)$ the maximum utilization of τ and we define it as: $U_{max}(\tau) = \max_i^n u_i$.

We also consider a set Π of m processors π_j indexed from 1 to m (*i.e.* $\Pi = \{\pi_1, \ldots, \pi_m\}$). We consider the processors as identical (*homogeneous case*). In other terms, each job is executed as the same rate on any processor.

3. SUSTAINABILITY

In order to analyze a given hard real-time system, a technique consists in highlighting the worst case scenario in which the worst response time could be met. Intuitively, this scenario has to occur when the utilization of the set of tasks is maximum. In other words, it has to occur when the execution requirement of each job is maximum and when the inter-arrival time of each task is minimum.

But it has been shown that some classes of algorithms are prone to scheduling anomalies [14, 13]. In particular, it has been shown that a fixed-priority restricted-migration job assignment is not predictable. Let $\mathcal{J}_k$ denote a job generated by a task τ_i. Let $\mathcal{S}_k$ denote the start time of $\mathcal{J}_k$ and $\mathcal{S}_k^-$ its minimum start time. In the same manner, let $\mathcal{F}_k$ denote the finish time of $\mathcal{J}_k$ and $\mathcal{F}_k^+$ its maximum finish time. The non-predictability of a scheduling algorithm is a scheduling anomaly which occurs when the execution time e^k of $\mathcal{J}_k$ is such that $e_k^- \leq e_k \leq e_k^+$ and (i) either $\mathcal{S}_k < \mathcal{S}_k^-$ or (ii) $\mathcal{F}_k > \mathcal{F}_k^+$.

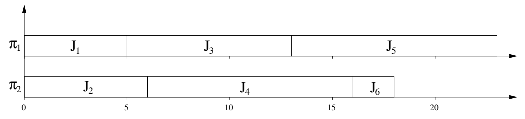
In Figure 1, we show an example from [14] illustrating this kind of scheduling anomaly. The Table 1(a) represents the arrival time r_k, the absolute deadline d_k and the execution requirement e_k of the job J_k for $1 \leq k \leq 6$. The priority of the jobs are given in decreasing order of their index (J_1 is the higher priority and J_6 is the lower priority). In this example, the execution requirement of J_2 can vary from 2 to 6. In Figure 1(b), respectively 1(c), J_2 runs for its maximum, respectively its minimum, execution requirement. We notice that all deadlines are met. Intuitively, we could conclude that no deadlines can be missed in all cases. But in Figure 1(d), J_2 runs for 3 time units and J_4 misses its deadline. Finally, in Figure 1(e), when J_2 runs for 5 time units, the finish time $\mathcal{F}_4$ of J_4 is minimum.

The predictability is a notion linked to the execution requirement of the jobs. It consists in guaranteeing that a scheduling algorithm can successfully schedule a set of jobs when the execution requirement of one or many jobs is reduced. The concept of sustainability comes from the notion of predictability but it is extended to the relaxation of other parameters as relative deadlines or arrival times.

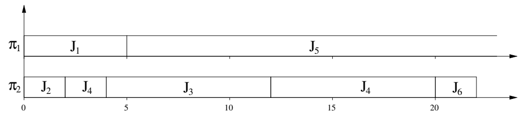
Definition 1 (Sustainable algorithm). *Let $\mathcal{A}$ denote a scheduling algorithm. Let τ denote any set of tasks that is $\mathcal{A}$-schedulable. Let $\mathcal{J}$ denote a collection of jobs generated by τ. Scheduling algorithm $\mathcal{A}$ is said to be sustainable if and only if $\mathcal{A}$ meets all deadlines when scheduling any collection of jobs obtained from $\mathcal{J}$ by changing the parameters*

job	r_k	d_k	$[e_k^-, e_k^+]$	job	r_k	d_k	$[e_k^-, e_k^+]$
J_1	0	10	$[5, 5]$	J_4	0	20	$[10, 10]$
J_2	0	10	$[2, 6]$	J_5	5	200	$[100, 100]$
J_3	4	15	$[8, 8]$	J_6	7	25	$[2, 2]$

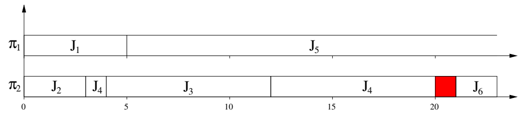
(a) Job parameters



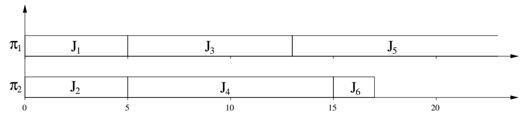
(b) $e_2 = 6$



(c) $e_2 = 2$



(d) $e_2 = 3$



(e) $e_2 = 5$

Figure 1: A example illustrating scheduling anomalies from [14].

of one or more individual jobs in any, some, or all the following ways: (i) decreased execution requirements; (ii) larger relative deadlines and (iii) latter arrival times.

In addition to the scheduling algorithms, the concept of sustainability can be used in the case of schedulability tests. In the same manner a scheduling policy can suffer from scheduling anomalies when job or task parameters are relaxed, a schedulability test can argue that a set of tasks is schedulable and invalidate this response if one or many of the scheduling parameters is relaxed.

DEFINITION 2 (SUSTAINABLE TEST). *Let $\mathcal{A}$ denote a scheduling algorithm and $\mathcal{F}$ a schedulability test for $\mathcal{A}$. Let τ denote any set of tasks that is $\mathcal{A}$-schedulable. Let $\mathcal{J}$ denote a collection of jobs generated by τ. Scheduling test $\mathcal{F}$ is said to be sustainable if and only if $\mathcal{A}$ meets all deadlines when scheduling any collection of jobs obtained from $\mathcal{J}$ by changing the parameters of one or more individual jobs in any, some, or all the following ways: (i) decreased execution requirements; (ii) larger relative deadlines and (iii) latter arrival times.*

In the case of uniprocessor scheduling, the schedulability tests have been studied in order to highlight which of them are sustainable [4, 6]. This work has been extended in the case of the multiprocessor scheduling [3].

4. SCHEDULING ALGORITHM

In Section 4.1 below, we describe the behavior of the fixed-priority restricted-migration scheduling algorithm as presented in [14]. In Section 4.2, we define *r-SP_wl*, a static-priority restricted-migration multiprocessor scheduling algorithm for scheduling periodic sets of tasks upon identical multiprocessor platforms. In Section 4.3, we prove that *r-SP_wl* is sustainable according to decreased execution time, later arrival times and larger relative deadlines. In Section 4.4, we review the sustainability of the schedulability tests of *r-SP_wl*.

4.1 Non-predictable algorithm

We have reminded in Section 3 that a fixed-priority restricted-migration scheduling algorithm is not predictable. By extension, a static-priority restricted-migration suffers from the same scheduling anomaly. In order to propose a predictable scheduling algorithm of this class, it is necessary to analyze the properties of the scheduling algorithm given in [14] to discover which property is responsible of the non-predictability. This scheduling algorithm is described by the Algorithm 1.

Algorithm 1: Fixed-priority restricted-migration scheduling algorithm from [14].

```
   Input: Ready job $J_k$
   Input: Global pending jobs queue $\mathbf{Q}^g$
   Input: Local pending jobs queues $\mathbf{Q} = \{\mathbf{Q}_1, \ldots, \mathbf{Q}_m\}$
 1 foreach $\pi_j \in \Pi$ do
 2     if $\pi_j$ is free then
 3         if there is a lower priority job $J_l$ running on $\pi_j$ then
 4             stop $J_l$;
 5             put $J_l$ in $\mathbf{Q}_j$;
 6         end
 7         start $J_k$ on $\pi_j$;
 8     else
 9         put $J_k$ in $\mathbf{Q}^g$;
10     end
11 end
```

This algorithm is designed to assign jobs on the multiprocessor platform when they are released only if a processor is available. In this last case, the assignment is postponed to a time at which a processor becomes available (the finish time of a higher priority job). The ready job J_k is tested to be assigned on one of the m available processors (Lines 1-11). If one of these processors is free – in the sense of no higher priority job is running on this processor – then J_k starts on π_j (Lines 2-7). If a lower priority job is already running, it is preempted and put in the π_j local queue (Lines 3-6). Otherwise, if no processor is free for J_k at its release time, its start is postponed and it is put in the global queue $\mathbf{Q}^g$ (Lines 8-9). For readability, we voluntarily omit the algorithm part corresponding to the finish time of a job. But when a job finishes its execution, the scheduler is called to check the pending local and global job queues in order to start the higher priority waiting job.

DEFINITION 3 (LOWER-PRIORITY-AGNOSTIC TEST). *A schedulability test is lower-priority-agnostic if for a given job J_k, no lower priority than J_k job has to be considered to decide the schedulability of J_k.*

As shown in Figure 1(d), the fixed-priority restricted-migration scheduling algorithm from [14] is *lower-priority-*

agnostic since J_3 is assigned on processor π_2 without consideration in $J4$ which will miss deadline because of lake of processor execution time.

Definition 4 (Processor-available assignment). *An assignment is processor-available if the assignment of a job J_k is made when a processor becomes available.*

As shown in Figures 1(b) and 1(c), the algorithm from [14] performs a *processor-available* assignment. The job J_3 released at time 4 is either assigned on processor π_1 at time 5 because no processor is available before or on processor π_2 at time 4 because of lower priority job runs and can be preempted.

Definition 5 (Work-conserving algorithm). *A multiprocessor scheduling algorithm is work-conserving if no processor is idled while there are active jobs.*

As explained above, when a job finishes its execution, the scheduler is executed in order to wake up a potential pending job from both the processor local queues and the global system queue. This behavior leads to produce a *work-conserving* scheduler.

We have shown that the fixed-priority restricted-migration scheduling algorithm from [14] is a *work-conserving* algorithm and that its job assignment is *processor-available* using a schedulability test which is *lower-priority-agnostic.*

4.2 Algorithm *r-SP_wl*

According to the previous cited properties, we now examine how to design a predictable static-priority restricted-migration scheduling algorithm. In a more general purpose, we extend this property of predictability to a sustainability one.

Algorithm 2: *r-SP_wl* scheduling algorithm.

```
   Input: Ready job $J_k$
   Input: Local pending jobs queues $\mathbf{Q} = \{\mathbf{Q}_1, \ldots, \mathbf{Q}_m\}$
1  foreach $\pi_j \in \Pi$ do
2      if $\pi_j$ is free then
3          if $J_k$ is the most priority on $\pi_j$ then
4              if there is a lower priority job $J_l$ running on $\pi_j$ then
5                  stop $J_l$;
6                  put $J_l$ in $\mathbf{Q}_j$;
7              end
8              start $J_k$ on $\pi_j$;
9          else
10             put $J_k$ in $\mathbf{Q}_j$;
11         end
12     else
13         return unschedulable;
14     end
15 end
```

This algorithm is designed to assign jobs on the multiprocessor platform at their release time even if no processor is available. In this last case, jobs are put in the local queue of the chosen processor. The ready job J_k is tested to be assigned on one of the m available processor (Lines 1-15). If one of these processors is free then J_k starts on π_j (Lines 2-11). With *r-SP_wl*, a processor π_j is considered free for the job J_k if all jobs $J_j \in J(\pi_j)'$ (with $J(\pi_j)' = J(\pi_j) \cup J_k$) have enough laxity $L_j(\pi_j, r_k)$ on processor π_j at their released time r_j to be scheduled. The laxity values are computed and maintained by the scheduler. If a lower priority job is already running, it is preempted and put in the π_j local queue (Lines 3-8). If no processor is free for job J_k at its release time, it is considered unschedulable. Like the algorithm presented in [14], the scheduler has to be executed at a time when a job finishes its execution but it has only to examine the local queue of the processor on which this job finishes. The laxity has also to be decreased for a job J_k at time $r_k + C_k$ in order to guarantee the predictability of the schedule.

Definition 6 (Laxity). *The laxity $L_k(\pi_j, t)$ of a job J_k on a processor π_j at time t is given by:*

$$L_k(\pi_j, t) = D_k - C_k - \sum_{J_h \in hp(\pi_j, J_k)} e^*_h(t)$$

where $e^*_h(t)$ denotes the remaining execution requirement of the job J_h at time t.

A static-priority schedulability test is in general *lower-priority-agnostic.* In uniprocessor, this behavior is completely justified because a higher priority job has to be scheduled before other jobs even if these last ones do not meet their deadline. But in the multiprocessor case, this behavior can lead to the case that the job is assigned on a processor on which a lower priority task could miss its deadline or migrate while another processor could discard this eventuality. In the case of a global scheduler, a *lower-priority-agnostic* test can lead to a more important number of migrations without impacting the schedulability. But in the case of restricted-migration approach, it can cause deadline misses.

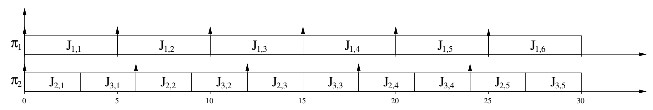
Definition 7 (Lower-priority-aware test). *A schedulability test is lower-priority-aware if for a given job J_k, lower priority than J_k jobs have to be considered to decide the schedulability of J_k.*

In the case of fixed-priority restricted-migration, one of the main causes of the predictability problem is the *processor-available* assignment. During the release time of a job and when a processor becomes free to admit it, many changes can occur in the scheduling scenario. These changes can lead to the job assigned on a completely different processor. At this point, it is impossible to predict the worst case behavior of this job since it is not allowed to migrate on another processor in order to continue its execution after a preemption by a higher priority job.

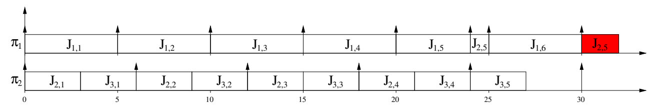
Definition 8 (Release-time assignment). *An assignment is release-time if the assignment of a job J_k is made when J_k becomes active (at its release time) instead of the time at which a processor becomes available.*

The algorithm *r-SP_wl* is driven by the laxity of the jobs at time t. We can notice that this value of laxity is computed using the WCET of the task which generates the job. If a job J_k finishes its execution earlier than this WCET, it can result in an idle time on the processor on which J_k was executed.

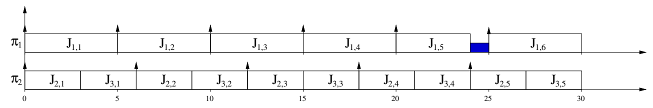
Definition 9 (Idle algorithm). *A multiprocessor scheduling algorithm is idle if at least one processor can be idled while there are active jobs.*



(a) Schedulable set of jobs.



(b) Unschedulable set of jobs without idle times.



(c) Schedulable set of jobs with idle times.

Figure 2: Impact of idle times in restricted-migration scheduling.

We illustrate in Figure 2 the problems which we can encounter with restricted-migration scheduling in the case of *work-conserving* algorithms and *lower-priority-agnostic* schedulability tests. The system depicted in this figure is generated by three tasks $\tau_1(5,5,5)$, $\tau_2(3,6,6)$ and $\tau_3(3,6,6)$ and scheduled among two processors. In Figure 2(a), we represent a set of jobs schedulable with respect to the static-priority restricted-migration scheduling. In Figure 2(b), we represent the same set of jobs but the job $J_{1,5}$ has an execution requirement of 4 instead of 5. In this case, the processor π_1 becomes idle at time 24. In this scenario, the job $J_{2,5}$ can be assigned to the processor π_1 and start its execution. But at time 25, the sixth job of τ_1 is released and preempts the job of τ_2 since τ_1 is higher priority. At this time, it is possible to know that $J_{2,5}$ will miss its deadline because it has not enough laxity to finish its execution. In Figure 2, the same set of jobs is depicted, but the *lower-priority-aware* test of *r-SP_wl* is used. Since the laxity of a job on a processor is updated at the time of its worst case finish time, the laxity of $J_{1,5}$ at time 24 is always 0 and therefore the laxity of $J_{2,5}$ is also 0. The processor π_1 can not be considered as available for the assignment of this last job. This behavior leads to produce idle times in scheduling. It is also important to notice that the *lower-priority-aware* schedulability test of *r-SP_wl* does not allow $J_{1,6}$ to be assigned on π_1 because the laxity of lower priority job $J_{2,6}$ would not be enough to permit it to finish its execution before its deadline.

We shown that *r-SP_wl* differs from the fixed-priority restricted-migration from [14] in three points. We now prove that it is sustainable.

4.3 Sustainability of *r-SP_wl*

In order to prove the sustainability of *r-SP_wl*, we must verify that it can successfully schedule a set of tasks according to the three relaxations of parameters which are (i) execution requirement decreasing (ii) relative deadline increasing (iii) arrival time increasing. We verify these points by the three following lemmas.

LEMMA 1. *Let τ denote any set of tasks that is schedulable by* r-SP_wl. *Let $\mathcal{J}$ denote a collection of jobs generated by τ The algorithm* r-SP_wl *meets all deadlines when scheduling any collection of jobs obtained from $\mathcal{J}$ by changing the parameters of one or more jobs by decreasing execution requirements.*

PROOF. The proof comes from a mathematical induction over the priority of the jobs. Let J be a set of n jobs. We denote J_i^- the job for which the execution requirement is such that $e_i^- < e_i$. We also denote $L_i^-(\pi_j,t) = D_i - C_i - \sum_{J_h^- \in hp(\pi_j, J_i^-)} e_j^{-*}(t)$ the laxity of J_i^- on the processor π_j at time t. The induction hypothesis is $\forall\ 1 \leq i \leq n,\ L_i^-(\pi_j,t) \geq L_i(\pi_j,t)$. The hypothesis is true at step one by definition of the laxity since J_i is the most priority job: $L_i^-(\pi_j,t) = L_i(\pi_j,t)$. We suppose the induction is true at step i. We verify that the relation $L_{i+1}^-(\pi_j,t) \geq L_{i+1}(\pi_j,t)$ is true for the job J_{i+1}. By contradiction:

$$L_{i+1}^-(\pi_j,t) < L_{i+1}(\pi_j,t) \qquad (1)$$

$$\Rightarrow \sum_{J_h^- \in hp(\pi_j, J_i^-)} e_j^{-*}(t) > \sum_{J_h \in hp(\pi_j, J_i)} e_j^*(t) \qquad (2)$$

Since π_j stay idle when higher priority jobs than J_i^- have finished their execution before their WCET, we obtain

$$\sum_{J_h^- \in hp(\pi_j, J_i^-)} e_j^{-*}(t) = \sum_{J_h \in hp(\pi_j, J_i)} e_j^*(t)$$

which is in contradiction with (2). The induction hypothesis follows. □

LEMMA 2. *Let τ denote any set of tasks that is schedulable by* r-SP_wl. *Let $\mathcal{J}$ denote a collection of jobs generated by τ The algorithm* r-SP_wl *meets all deadlines when scheduling any collection of jobs obtained from $\mathcal{J}$ by changing the parameters of one or more jobs by enlarging relative deadlines.*

PROOF. The scheduling decisions are made considering the laxity at time t of the jobs. The deadline of the jobs is not used to compute this laxity then scheduling decision can not be changed if a deadline is enlarged. □

LEMMA 3. *Let τ denote any set of tasks that is schedulable by* r-SP_wl. *Let $\mathcal{J}$ denote a collection of jobs generated by τ The algorithm* r-SP_wl *meets all deadlines when scheduling any collection of jobs obtained from $\mathcal{J}$ by changing the parameters of one or more jobs by enlarging arrival times.*

PROOF. The considered tasks are periodic then the arrival time of a job can not be increased. □

PROPOSITION 1. *The static-priority restricted-migration scheduling algorithm* r-SP_wl *is sustainable with respect to WCET, relative deadlines and period.*

PROOF. The proof is made from Lemmas 1, 2 and 3. □

4.4 Sustainability of *r-SP_wl* tests

In order to decide the schedulability of a set of tasks with *r-SP_wl*, we proposed a sufficient schedulability test based on the notion of *LOAD*. *LOAD* at level k represents the maximum processor demand of the jobs of priority greater than and equal to k at any time t.

THEOREM 1 (FROM [8]). *Any set τ of periodic tasks satisfying:*

$$\forall k : 1 \leq k \leq n : \left[LOAD(k) \leq \frac{1 + (m-1)U_{min}(k)}{1 + 2(D_{max}(k)/D_k)} \right]$$

is r-SP_wl *schedulable upon identical multiprocessor platforms where* $U_{min}(k)$ *is the minimum utilization and* $D_{max}(k)$ *the maximum deadline of the tasks of priority from* 1 *to* k.

The sufficient schedulability condition given by Theorem 1 is not sustainable. In order to decide schedulability of a set of tasks without sustainability problem, it is necessary to consider the schedule over the feasibility interval given by $[X_1, S_n + P]$ where X_n is given by $X_n = S_n$ and $X_i = O_i + \left\lfloor \frac{X_{i+1} - O_i}{T_i} \right\rfloor T_i$ and where S_n is given by $S_1 = O_1$ and $S_i = \max(O_i, O_i + \left\lceil \frac{S_{i-1} - O_i}{T_i} \right\rceil T_i)$. P denotes the hyperperiod of the set of tasks and O_i the offset of the task τ_i.

5. CONCLUSION

In the case of embedded real-time multiprocessor scheduling, the restricted-migration scheduling is an interesting approach. But it has been shown that it suffers from sustainability problems in the case of fixed-priority. In this paper, we review the notion of sustainability and we analyze a non-predictable algorithm in order to propose a sustainable alternative. We highlight the properties of *lower-priority-agnostic* test, *processor-available* assignment and *work-conserving* algorithm which are responsible of the non-predictability. We propose for our algorithm *r-SP_wl* the properties of *lower-priority-aware* test, *release-time* assignment and *idle* algorithm. We prove that it does not suffer from scheduling anomaly with the model of tasks we consider.

In a previous work, we proposed a sufficient schedulability test for *r-SP_wl*. Unfortunately, this test is not sustainable and we intend to develop a sustainable one in a future work.

6. REFERENCES

[1] B. Andersson, S. K. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, pages 193–202, London, UK, December 2001. IEEE Computer Society.

[2] T. P. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Technical Report TR-050601, Florida State University, Tallahassee, FL, USA, July 2005.

[3] T. P. Baker and S. K. Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *Proceedings of the 21st Euromicro Conference on Real-time Systems (ECRTS)*, pages 141–150, Dublin, Irland, July 2009. IEEE Computer Society.

[4] S. K. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS)*, pages 159–168, Rio de Janeiro, Brazil, December 2006. IEEE Computer Society.

[5] S. K. Baruah and J. Carpenter. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *Proceedings of the 15th Euromicro Conference on Real-time Systems (ECRTS)*, pages 195–202, Porto, Portugal, July 2003. IEEE Computer Society.

[6] A. Burns and S. K. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering (JCSE)*, 2(1):74–97, March 2008.

[7] J. Carpenter, S. H. Funk, P. Holman, A. Srinivasan, J. H. Anderson, and S. K. Baruah. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms, pages 30–1 – 30–19. Chapman and Hall/CRC, Boca Raton, Florida, 2004.

[8] F. Fauberteau, S. Midonnet, and L. George. Laxity-based restricted-migration scheduling. In *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Toulouse, France, September 2011. IEEE Computer Society.

[9] N. W. Fisher. *The Multiprocessor Real-Time Scheduling of General Task Systems.* PhD thesis, University of North Carolina at Chapel Hill, 2007.

[10] S. H. Funk. *EDF Scheduling on Heterogeneous Multiprocessors.* PhD thesis, University of North Carolina at Chapel Hill, 2004.

[11] L. George, P. Courbin, and Y. Sorel. Job vs. portioned partitioning for the earliest deadline first semi-partitioned scheduling. *Journal of Systems Architecture*, 57(5):518–535, May 2011.

[12] J. Goossens, S. H. Funk, and S. K. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, September 2003.

[13] R. Ha. *Validating timing constraints in multiprocessor and distributed real-time systems.* PhD thesis, University of Illinois, Dept. of Computer Science, Urbana-Champaign, IL, USA, 1995.

[14] R. Ha and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS)*, pages 162–171, Pozman, Poland, June 1994. IEEE Computer Society.

[15] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, December 1982.

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

[17] J. M. López, M. García, J. L. Díaz, and D. F. García. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proceedings of the 12th Euromicro Conference on Real-time Systems (ECRTS)*, pages 25–33, Stockholm , Sweden, June 2000. IEEE Computer Society.

[18] A. K.-L. Mok. *Fundamental Design Problems of Distributed Systems for Hard-Real-Time Environments.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, May 1983.

[19] D.-I. Oh and T. P. Baker. Utilization bounds for n -processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2):183–192, September 1998.