Laxity-Based Restricted-Migration Scheduling

Frédéric Fauberteau, Serge Midonnet, Laurent George

Laxity-Based Restricted-Migration Scheduling

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

Laurent George
ECE
LACSC
37, quai de Grenelle
75015, Paris, France
lgeorge@ieee.org

Abstract

We focus on the real-time multiprocessor scheduling of periodic tasksets. We propose a new static priority scheduling algorithm based on the restricted-migration approach. Restricted-migration approach is a global scheduling approach for which the number of migrations is bounded just by one migration per job at most. Our algorithm uses the laxity of already admitted jobs to decide the admission of newly arrived jobs. We prove that this algorithm is predictable. We give a feasible interval and we propose a utilization bound for this algorithm. We also compare our algorithm to other global algorithms in terms of schedulability by simulations.

1. Introduction

One of the most studied model of recurring real-time tasks is the periodic task model of Liu and Layland [1]. In the periodic model, a periodic task is characterized by a Worst Case Execution Time (WCET) and a period. Each job of a recurring task must be completed before its absolute deadline which is equal to its activation time plus the relative deadline of the task. Three levels of constraint on the deadline of tasks are studied in the literature: (i) *implicit deadlines* for which the deadline of a task is equal to its period, (ii) *constrained deadlines* for which the deadline of a task is less than or equal to its period and (iii) *arbitrary deadlines* for which there are no relations between the deadline of a task and its period. A recurring task is also characterized by an offset. The offset of a task corresponds to the time before which the first job of this task is activated. This research focus on the scheduling of periodic tasks with constrained deadlines and offsets.

A priority-driven preemptive scheduler schedules a set of tasks by executing the highest priority active job and by preempting the actual executed job if the processor is not idle. There are three main classes of priority algorithms: (i) *static priority algorithms* (or *fixed tasks priority* algorithms), (ii) *fixed job priority algorithms* and (iii) *dynamic priority algorithms*. Static priority algorithms assign a priority to each task and each job of a task inherits the priority of this task. A job keeps the same priority during its lifetime (*e.g. Rate Monotonic* [1], *Deadline Monotonic* [2]). Fixed job priority algorithms assign a priority to each job and it keeps it during its lifetime (*e.g. Earliest Deadline First* (EDF) [1]). Dynamic priority algorithms update the priority of jobs during their lifetimes (*e.g. Least Laxity First* [3]). In this paper, we consider static priority scheduling of a set of periodic tasks.

In multiprocessor scheduling, there are two main approaches: (i) *partitioned scheduling* and (ii) *global scheduling*. In partitioned scheduling, the tasks are statically assigned to a processor and each job of a task is scheduled on the processor on which this task has been assigned. The interprocessor migrations are not allowed with this approach. In global scheduling, there is no assignment of task and unrestricted migrations of the jobs are allowed. In uniprocessor scheduling, EDF is an optimal algorithm for the implicit deadlines case. With this algorithm, a processor can schedule as many tasks as its capacity enables it (*i.e.* the sum of the utilization of tasks can be equal to 1). But in the multiprocessor case, EDF is not an optimal priority algorithm for global scheduling. There are global algorithms with dynamic priority which are optimal (*e.g. PF* [4], PD^2 [5]). But these algorithms can lead to many context switches (*i.e.* preemptions and migrations). Consequently, the scheduling approaches with static priority and fixed job priority are still

actively studied.

Global and partitioned scheduling approaches are incomparable in static priority and fixed job priority [6] (*i.e.* there are tasksets that are schedulable using global algorithms that partitioned algorithms cannot schedule, and *vice versa*). Although global scheduling does not outperform partitioned scheduling, a global scheduler is more suitable for taking good scheduling decisions. While the scheduling decisions of a partitioned scheduler are taken *a priori*, a global scheduler decides to assign tasks dynamically and can therefore take into account the state of all the processors. But this behavior can lead to an important number of migrations (*e.g.* DP-Wrap algorithm produces at most $m-1$ migrations by time slice [7]). The cost of migrations can represent a considerable overhead. Therefore we focus on a third approach for multiprocessor scheduling. This approach is the *restricted-migration scheduling* approach for which the tasks are allowed to migrate just one time per job at most. For instance, a job of a task τ_2 can not migrate at job of τ_1 arrival time (with τ_1 more priority than τ_2).

In this paper, we have proposed a new scheduling algorithm for the restricted-migration approach. This approach is known to have scheduling anomalies associated to WCETs (a scheduling valid when tasks experience their WCET can become non feasible when the execution duration of a task is less than its WCET). Our scheduler makes scheduling decisions by considering the laxity of already assigned jobs. The laxity of a job is the duration between its finish time and its absolute deadline. By taking into account the laxity of the jobs, our algorithm assigns the newly activated jobs to a processor with the guaranty that no already assigned jobs miss their deadline on this processor. We show that our algorithm is predictable and does not lead to scheduling anomalies associated to WCETs. We propose a feasibility condition for our algorithm that we compare to a well known global EDF feasibility condition. We then study by simulations the performances of our scheduling algorithm.

1.1. Related work

Baruah and Carpenter have proposed a restricted-migration with fixed job priority algorithm to schedule periodic tasksets (*r-EDF*) [8]. This algorithm has been designed to deal with hard real-time task systems (no deadline miss is allowed). Anderson, Bud and Devi have proposed the *EDF-fm* algorithm [9]. This algorithm has been designed to schedule soft real-time task systems (some deadlines can be missed). To the best of our knowledge, this scheduling approach has not been very studied in static priority.

For a long time, the laxity is a parameter used for scheduling decisions. One of the well-known example is LLF [3], which gives the highest priority to the task with the lowest value of laxity. But LLF suffers from high context switching overhead and Lee has proposed the scheduling algorithm *Earliest Deadline until Zero Laxity* (EDZL) [10]. EDZL is based on EDF but short-circuits it when a task has no laxity to give the highest priority to this task. Recently, Davis and Burns have proposed a similar algorithm called *Fixed Priority until Zero Laxity* (FPZL) [11]. This algorithm uses a *fixed task priority* instead of EDF.

1.2. Remainder

The remainder of this paper is organized as follows: in Section 2, we describe the notations used in this paper. In Section 3, we present our restricted-migration scheduling algorithm. In Section 4, we discuss the predictability of our proposed algorithm. In Section 5, we give a sufficient feasibility condition based on the load(τ) function. In Section 6, we compare our proposed algorithm with a global and a restricted-migration scheduler, and finally, we conclude in Section 7.

2. Terminology

We consider a taskset τ composed of n periodic tasks (*i.e.* $\tau = \{\tau_1, \ldots, \tau_n\}$). The tasks are considered in decreasing order of their priorities (*i.e.* τ_1 is the highest priority task and τ_n is the lowest priority task). Each task τ_i is characterized by:

- the activation time of its first job (*i.e.* its offset) denoted O_i,
- its WCET denoted C_i,
- its relative deadline denoted D_i,
- its period denoted T_i.

We denote U_i the utilization of the task τ_i defined by $U_i = \frac{C_i}{T_i}$. A periodic task is a recurrence of jobs and the k^{th} job $J_{i,k}$ of the task τ_i is characterized by:

- its activation time denoted $r_{i,k}$ and defined by $r_{i,k} = O_i + kT_i$,
- its execution requirement denoted $e_{i,k}$,
- its absolute deadline denoted $d_{i,k}$ and defined by $d_{i,k} = r_{i,k} + D_i$.

By definition, $e_{i,k} = C_i$. We denote $e_{i,k}(t)$ the remaining execution time of job $J_{i,k}$ at time t and by definition $e_{i,k} = e_{i,k}(r_{i,k})$. To improve readability, we denote J_i a job of τ_i when we do not refer to a

particular job of τ_i. J_i is characterized by (r_i, e_i, d_i). The priorities are indexed from 1 to n. We denote $hp(J_i, \pi_j)$ the set of higher priority jobs than J_i on the processor π_j and $lp(J_i, \pi_j)$ the set of lower priority jobs than J_i on the processor π_j.

We consider that τ is scheduled on a set Π of m identical processors (*i.e.* $\Pi = \{\pi_i, \ldots, \pi_m\}$). We denote $J(\pi_m)$ the set of jobs which has been assigned on the processor π_m.

3. Scheduling algorithm

Contrary to LLF, EDZL or FPZL, our algorithm is a pure static-priority algorithm. The priority of a task (and by extension of all the jobs of this task) is decided just one time. The laxity is used to take the scheduling decisions at jobs arrival times.

We now describe the behavior of a restricted-migration scheduler on a system consisting of m processors. The activated jobs are put in a global queue. When a processor π_j becomes idle respectively to the highest priority job J_i of the global queue, J_i is started on π_j. The processor π_j is idle respectively to the job J_i if either no jobs are executed on π_j or a job of lower priority than J_i is executed on π_j. If J_k $(k > i)$ is executed on π_j before the activation of J_i, J_k is preempted and moved to the local queue of π_j. J_k can be restarted only if π_j becomes idle respectively to this job.

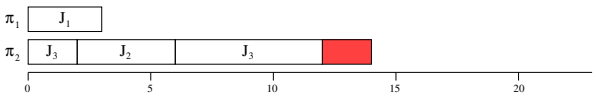


Figure 1. Restricted-migration scheduling.

In Figure 1, we represent an example taskset composed of 3 jobs: J_1 $(0, 3, 5)$, J_2 $(2, 4, 8)$ and J_3 $(0, 10, 12)$. At time 0, J_1 is assigned on the idle processor π_1 and J_3 is assigned on the idle processor π_2. At time 2, J_2 is activated and the only idle processor respectively to J_2 is π_2. Then J_2 preempts J_3 and is executed for 4 time on π_2. Unfortunately, J_3 misses its deadline at time 12 with 2 remaining execution time.

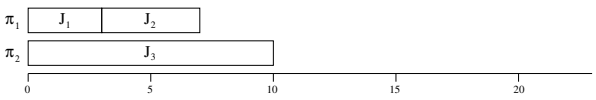


Figure 2. Laxity-aware restricted-migration scheduling.

We notice that these 3 jobs can be successfully scheduled as shown in Figure 2. Actually, J_2 must be assigned at time 2 on the non-idle processor π_1. From this observation, we have designed a restricted-migration scheduling algorithm (*r-SP_wl*) using the laxity to decide the admission of jobs.

Definition 1 (Laxity)**.** *The laxity $L^j_{i,k}(t)$ of the job $J_{i,k}$ on the processor π_j at time t is the duration between the job ending and its deadline. $L^j_{i,k}(t)$ is defined on the time interval $[r_{i,k}, d_{i,k}]$.*

The laxity of the job J_i on the processor π_j is defined at time r_i by:

$$L^j_i(r_i) = d_i - e_i - \sum_{J_j \in hp(J_i, \pi_j)} e_j(r_i) \qquad (1)$$

Contrary to the original restricted-migration scheduling algorithm, our algorithm does not use global queue. Only a local queue per processor is needed to maintain the pending preempted jobs. The jobs are assigned on a processor at time of their activations. A job $J_{i,k}$ can be assigned on a processor π_j at time $r_{i,k}$ if and only if these 2 conditions are verified:

1) the laxity of the job $J_{i,k}$ on π_j is greater than or equal to 0: $L^j_{i,k}(r_{i,k}) \geq 0$ with $L^j_{i,k}(r_{i,k})$ given by Equation (1),
2) the laxity of the lower priority jobs than $J_{i,k}$ is greater than or equal to 0.

The Condition 2) is given by:

$$\forall J_l \in lp(J_{i,k}, \pi_j),\ L^j_l(r_{i,k}) - e_{i,k} \geq 0 \qquad (2)$$

The condition given by Equation (2) insures that no lower priority jobs previously assigned on the processor π_j miss their deadlines. Therefore the scenario depicted in Figure 1 can not occur.

Our scheduling algorithm must maintain the values of jobs laxity in order to take scheduling decision at activation time of jobs. A data structure per processor is needed to store these values of laxity. We define the concept of processor laxity. The laxity of the processor π_j is defined as the laxity of the job which has the minimum value of laxity on π_j. Our algorithm works as follow:

1) when a job J_i is activated, the remaining time of all the current running jobs is updated,
2) we consider the processors of Π taken in decreasing order of their laxity. J_i is assigned on the first processor π_j for which the Conditions 1) and 2) are verified (the processor with the maximum value of processor laxity on which J_i can be assigned),
3) a new entry in the structure is added to store its value of laxity computed by Equation (1),

4) the laxity of jobs of $lp(J_i, \pi_j)$ is decreased by e_i,
5) when J_i finishes at time t, its corresponding entry in the structure is removed. If J_i finishes prior to e_i execution time, the value of laxity of all lower priority jobs is increased by the remaining execution time $e_i(t)$ of J_i.

Our algorithm runs with a linear time complexity since there are just one activated job per task at most. Likewise the space complexity is also linear for the same reason.

4. Predictability

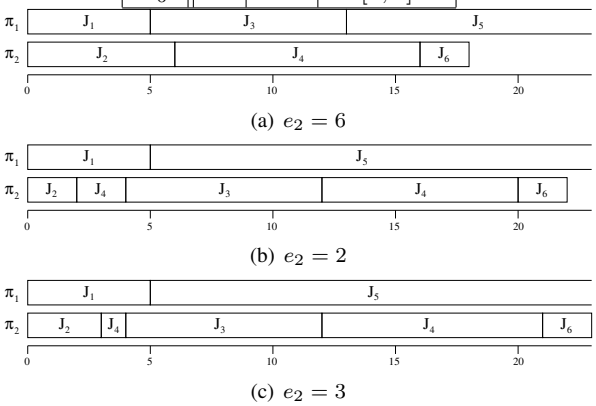
In this section, we consider the schedulability of a set J of jobs (*i.e.* $J = \{J_1, \ldots, J_l\}$). We consider this set of jobs ordered by decreasing priorities (*i.e.* $J_1 > \ldots > J_l$). The execution time e_i of a job J_i is a value in the time interval $[e_i^-, e_i^+]$. We denote J_i^- the job characterized by (r_i, e_i^-, d_i) and J_i^+ the job characterized by (r_i, e_i^+, d_i). We denote $J^{(k)}$ the set $\{J_1, \ldots, J_k\}$. We also denote $J_-^{(k)} = \{J_1^-, \ldots, J_k^-\}$ and $J_+^{(k)} = \{J_1^+, \ldots, J_k^+\}$. $S(J)$ denotes the start time of the lowest priority jobs of the set J. $F(J)$ denotes the ending time of the lowest priority jobs of the set J.

Definition 2 (Predictable algorithm)**.** *An scheduling algorithm is predictable if and only if both the start times and ending times are predictable.*

The start times are predictable implies $S(J_-^{(k)}) \leq S(J^{(k)}) \leq S(J_+^{(k)})$ such that $1 \leq k \leq l$. The ending times are predictable implies $F(J_-^{(k)}) \leq F(J^{(k)}) \leq F(J_+^{(k)})$ such that $1 \leq k \leq l$.

Ha and Liu have shown that a restricted-migration scheduler is not predictable in static-priority [12]. In Figure 3, we show the example they presented to illustrate this property. In Figure 3(a) and 3(b), the schedules arise when the execution time of J_2 has the maximum value 6 and the minimum value 2. We would conclude that these jobs always meet their deadlines. Unfortunately, this assumption is false as shown in Figure 3(c). We can then suspect *r-SP_wl* of being prone to anomalies of WCET reduction. In the example of the Figure 3, we notice that the standard restricted-migration scheduler take decisions independently of the already allocated jobs. The scenario depicted in the Figure 3(c) can not occur with our algorithm because *r-SP_wl* does not allow J_3 to be assigned on π_2. In Figure 4, we represent the schedule obtained from *r-SP_wl* for the jobs from the previous example. Contrary to the standard restricted-migration scheduler which keeps J_4

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



(a) $e_2 = 6$

(b) $e_2 = 2$

(c) $e_2 = 3$

Figure 3. Example from Ha and Liu [12]

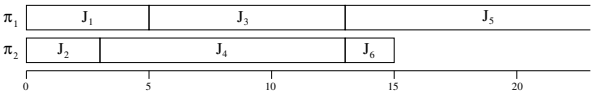
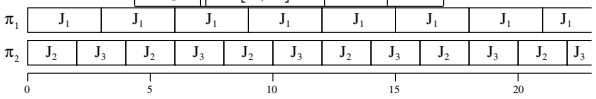


Figure 4. Jobs scheduled with *r-SP_wl*

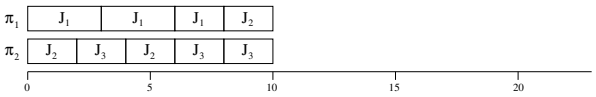
in the global queue until a processor becomes idle, our scheduler assigns J_4 on the processor π_1 at time of its activation (π_1 has the highest processor laxity at time 0). Then, J_3 can not be assigned on π_1 anymore because J_4 has not enough laxity. Our algorithm does not suffer from the anomalies presented by Ha and Liu.

However, our algorithm is not predictable as shown in Figure 5. The Figure 5(a) represents 3 periodic

job	$[e_i^-, e_i^+]$	D_i	T_i
J_1	$[2, 3]$	3	3
J_2	$[2, 2]$	4	4
J_3	$[2, 2]$	4	4



(a) $e_1 = 3$



(b) $e_1 = 3$, $e_{1,3} = 2$

Figure 5. Anomalies for *r-SP_wl*

tasks which are executed for their maximum execution time. No deadlines are missed for this scenario. But in Figure 5(b), the job $J_{1,3}$ is executed for its minimum execution time. Then at time 8, the processor π_1 becomes idle and J_2 can be assigned on it. J_3 is always assigned on π_2 and is started at time 8. At time 9, J_1 is reactivated but can be assigned neither on π_1 nor π_2 unless the deadline of either J_1 or J_2 be missed.

In order to avoid these anomalies, we propose an extension for *r-SP_wl* by adding a second data structure to maintain the laxity values. This second data is used in parallel of the first one and the execution time e_i is replaced by the WCET C_i in Equation (1). If a job finishes prior to C_i, the laxity value of the lower priority jobs is not increased in the second structure. Thus, our scheduler maintains "virtual" value of laxity in parallel of the exact values. These virtual values of laxity depend of the WCET of tasks and do not suffer from variations of the execution time.

Proposition 1. *The algorithm* r-SP_wl *is a predictable algorithm.*

Proof: It is sufficient to notice that the scheduler makes the assignment decision independently of the actual execution time of jobs but dependently of the WCET of tasks. □

5. Schedulability conditions

5.1. Necessary and sufficient condition

In order to decide if a taskset is schedulable, we propose a necessary and sufficient schedulability condition for *r-SP_wl*. This condition consists in scheduling the taskset over a feasibility interval.

Definition 3 (Feasibility interval)**.** *Let τ be a taskset and Π be a set of processors. A feasibility interval is a finite time interval such that if no deadline is missed in this time interval, then τ is schedulable on* Π.

An asynchronous activation scenario for a periodic taskset τ is the set $\{O_1, \ldots, O_n\}$ such that $\exists i, 1 \leq i \leq n, \quad O_i \neq 0$. In [13], Cucu and Goossens have given a feasibility interval for a taskset with asynchronous activation scenario. This time interval is built from the hypothesis that the scheduling of a taskset with asynchronous scenario is periodic from time S_n with the period P. The period P is equal to least common multiple of the task periods of τ. S_n is the time from which at least one job of all tasks has been activated. It is recursively defined by:

- $S_1 = 0_1$;
- $S_i = \max(O_i, O_i + \lceil \frac{S_{i-1} - O_i}{T_i} \rceil T_i)$.

The time interval starts from X_1. $[X_1, S_n]$ includes the activations of the jobs which have their deadline after time S_n. X_1 is recursively defined by:

- $X_n = S_n$;
- $X_i = O_i + \lfloor \frac{X_{i+1} - O_i}{T_i} \rfloor T_i$.

Theorem 1 (Corollary 10 [13])**.** *For any schedulable taskset τ on m processors, $[X_1, S_n + P]$ is a feasibility interval.*

The schedule produced by our algorithm *r-SP_wl* must be periodic if we want to use this feasibility interval given by Theorem 1.

Proposition 2. *The scheduling of any taskset τ produced by algorithm r-SP_wl is periodic from S_n with period P.*

Proof: The proof is the same as the one given for the Theorem 9 in [13]. □

Proposition 3. *The time interval $[X_1, S_n + P]$ is a feasibility interval to decide the schedulability of a taskset scheduled by* r-SP_wl.

Proof: The proof follows from Theorem 1 and Proposition 2. □

We must identify the worst case activation scenario to decide the schedulability of a taskset τ on a set of processor Π.

Definition 4 (Worst case activation scenario)**.** *An activation scenario characterized by $\{O_1, \ldots, O_n\}$ for a taskset τ of n tasks is the worst case activation scenario if and only if τ is schedulable (*resp. *unschedulable) implies τ is schedulable (*resp. *unschedulable) for any activation scenario.*

In static-priority uniprocessor scheduling, the worst case activation scenario is the synchronous scenario. The synchronous scenario is the scenario for which $\forall i \in 1 \leq i \leq n, \quad O_i = 0$. In the multiprocessor case, the worst case scenario is not necessarily the synchronous scenario.

Proposition 4. *The worst case activation scenario for a taskset τ is not necessarily the synchronous scenario.*

Proof: We consider the taskset τ consisting of 6 tasks: τ_1 $(0, 6, 6, 14)$, τ_2 $(0, 7, 7, 12)$, τ_3 $(0, 1, 11, 16)$, τ_4 $(0, 7, 27, 57)$, τ_5 $(0, 1, 62, 67)$ and τ_6 $(0, 21, 81, 88)$. This synchronous taskset is schedulable with *r-SP_wl* since no deadline has been missed during the simulation of this scheduling over the interval $[0, 4705008]$. However, the taskset obtained from τ with $0_3 = 1$ is not schedulable and τ_6 misses its deadline at time 3329384. □

Hence all possible scenarios have to be considered to identify the worst case activation scenario. Goossens has shown that just $\frac{\prod_{i=1}^{n} T_i}{P}$ are non-equivalent activation scenarios [14]. To decide the schedulability of a taskset τ on a set of Π processors, all the non-equivalent scenarios must be studied on their time interval $[X_1, S_n + P]$.

5.2. Utilization Bound

The necessary and sufficient schedulability condition for the algorithm *r-SP_wl* is exponential regarding the number of tasks. In the worst case, periods with different prime number values exponentially increase the least common multiple P. Therefore we propose a utilization bound to decide the schedulability of a taskset τ with this algorithm.

This bound is based on *LOAD* and is inspired by the bound proposed by Baruah and Fisher [15].

Definition 5 (DBF)**.** *For any time interval of length t, the $DBF(\tau_i, t)$ of a task τ_i bound the execution time of the jobs of τ_i which both are activated and have their deadline in this interval.*

Baruah, Mok and Rosier have shown that $DBF(\tau_i, t) = \max\left(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1)C_i\right)$.

Definition 6 (LOAD)**.** *For any k, the parameter* LOAD *is defined by:*

$$LOAD(k) = \max_{t>0} \left(\frac{\sum_{j=1}^{k} DBF(\tau_j, t)}{t} \right)$$

We consider a taskset τ for which the priorities are assigned by a static-priority algorithm. We consider a scenario where a job of the task τ_k misses its deadline. We denote t_a the activation time of this job and t_d the time of the deadline miss. We notice that:

$$t_d - t_a = D_k \tag{3}$$

We consider $W(t_a)$ the execution time of the jobs which have a deadline $\leq t_d$ and are executed over the time interval $[t_a, t_d)$. Our algorithm assigns the job τ_k on a processor π_j because it had sufficiently laxity to be executed. But a higher priority job has been activated prior to t_d which implies a deadline miss for job J_k. This processor has executed jobs for a duration $t_d - t_a$ (it has not been idle). The only hypothesis we can do on the other processors is these processors were not idle when jobs have been admitted on π_j. If a processor is idle, its minimum laxity value is considered as infinite and a job might have been admitted on it. We make the worst hypothesis that other processors than π_j run the jobs which have the lowest utilization. We obtain:

$$W(t_a) > (t_d - t_a) + (m-1)(t_d - t_a)U_{min}(k) \tag{4}$$

where $U_{min}(k) = \min_{j=1}^{k} U_j$.

Since $W(t_a)$ is the execution demand on the time interval $[t_a, t_d)$, all jobs which contribute to $W(t_a)$ must have a scheduling window which intersects $[t_a, t_d)$. Then we consider the jobs which are activated after $t_a - D_{max}(k)$ and have their deadline prior to $t_d + D_{max}(k)$ where $D_{max}(k) = \max_{j=1}^{k} D_j$. All these jobs have their activation and their deadline in the time interval of length $(2D_{max}(k) + (t_d - t_a))$. By definition of the *LOAD* parameter, we have:

$$W(t_a) \leq (t_d - t_a + 2D_{max}(k)) \times LOAD(k)$$

By Inequality 4, we have:

$$\begin{aligned}
&(t_d - t_a) + (m-1)(t_d - t_a)U_{min}(k) < \\
&(t_d - t_a + 2D_{max}(k)) \times LOAD(k) \\
\equiv &(t_d - t_a)(1 + (m-1)U_{min}(k)) < \\
&(t_d - t_a + 2D_{max}(k)) \times LOAD(k) \\
\equiv &1 + (m-1)U_{min}(k) < \\
&\left(1 + 2\frac{D_{max}(k)}{t_d - t_a}\right) \times LOAD(k)
\end{aligned}$$

By Inequality 3, we have:

$$1 + (m-1)U_{min}(k) < \left(1 + 2\frac{D_{max}(k)}{D_k}\right) \times LOAD(k)$$

This condition is a necessarily condition for a job missing its deadline. The negation of this condition produces a sufficient condition of schedulability.

Proposition 5. *A sufficient schedulability condition for a taskset τ of n tasks such that τ is schedulable with r-SP_wl is given by:*

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

6. Evaluation

In order to compare our proposed algorithm with existing approaches, we performed some simulations. The simulation protocol we use is extracted from the literature [16]. For each utilization value $U_i = i \times m$ where $i \in [0.025, 0.05, \ldots, 0.975]$, we randomly generate a set of tasksets for which the utilization $U(\tau) = U_i$. The generation of the taskset is based on the algorithm *UUnifast* [17]. This algorithm has been initially proposed in order to randomly generate

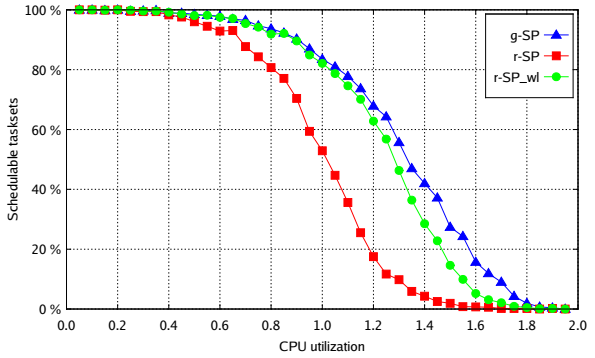


Figure 6. Comparison of schedulability.

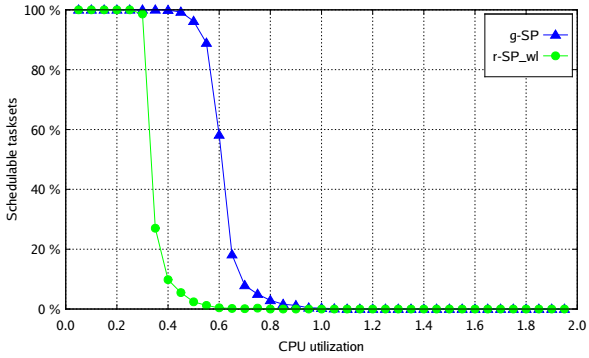


Figure 7. Comparison of utilization bound.

taskset for uniprocessor systems. It has been extended to *UUniFast-Discard* [16] for the multiprocessor case.

We show in Figure 6 a comparison of schedulability for three different algorithms in static priority. The first one is a global scheduler, the second one is a standard restricted-migration scheduler and finally, the last one is our modified restricted-migration scheduler. The standard static-priority restricted-migration has been implemented as follows. The set of processors is sorted (using a priority queue) in increasing order of the priority of the active job (idle processors are put in head of the list). A job is put in a priority queue at its arrival time. A processor π_j is free relative to job J_i if π_j is idle or its active job is lower priority than J_i. The highest priority jobs are dequeued and put on the first free processor of the processors queue while there are ready active job and free processors.

In order to get a good intuition about the schedulability of these algorithms, we simulated the execution of the randomly generated tasksets over their feasibility intervals. Since the size of the feasibility intervals is exponential in the number of tasks, we bounded to 6 the number of tasks per taskset and to 1000 the number of tasksets per value of utilization. For the same reasons, we restrict the number of processors to 2. We notice that our proposed algorithm gives results which are much closer to those of a global scheduler than those of a standard restricted-migration scheduler. This result is interesting in the sense that the number of interprocessor migrations is bounded by just one per job at most.

We show in Figure 7 the comparison between the utilization bound presented in [15] and the one we proposed in Section 5.2. These two utilization bounds are based on the load bound function. Our implementation of this function is based on the one described in [18]. Unfortunately, the computation of the exact load bound function is done in exponential time and chose the computation based on the pseudo-polynomial approximation scheme with a precision given by $\epsilon = 10^{-6}$. We notice that the utilization bound of our algorithm is not as good as the one of global scheduler. As we can see in [19], in the current state-of-the-art of schedulability conditions, no utilization bounds give efficient results.

global			
restricted-migration	this research		
partitioned			
	static	fixed job	dynamic

Table 1. Classification of algorithms for multiprocessor scheduling of periodic tasks.

7. Conclusion

We distinguish among three different categories of priority algorithms. We also distinguish among three different scheduling approaches concerning the amount of interprocessor migration allowed. There are $3 \times 3 = 9$ categories of scheduling algorithms which we represent in Table 1. We have focused on the restricted-migration approach in the static priority case and we have proposed a new algorithm based on the laxity. This algorithm is predictable and we have proposed a necessary and sufficient schedulability condition as well as a utilization bound. We have shown by simulation that our algorithm can schedule almost tasksets as much as a global scheduling algorithm. In order to extend this work, we intend to improve the proposed utilization bound. Since our scheduler dynamically maintains the value of laxity, we also intend to propose a slack-stealer algorithm which can be used in conjunction with *r-SP_wl*.

References

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

[2] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, December 1982.

[3] A. K.-L. Mok, "Fundamental design problems of distributed systems for hard-real-time environments," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, May 1983.

[4] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, June 1996.

[5] J. H. Anderson and A. Srinivasan, "Early-release fair scheduling," in *Proceedings of the 12th Euromicro Conference on Real-time Systems (ECRTS)*. Stockholm, Sweden: IEEE Computer Society, June 2000, p. 35–43.

[6] S. K. Baruah, "Techniques for multiprocessor global schedulability analysis," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS)*. Tucson, Arizona, USA: IEEE Computer Society, December 2007, pp. 119–128.

[7] G. Levin, S. H. Funk, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *Proceedings of the 22nd Euromicro Conference on Real-time Systems (ECRTS)*. Brussels, Belgium: IEEE Computer Society, July 2010, pp. 3–13.

[8] 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)*. Porto, Portugal: IEEE Computer Society, July 2003, pp. 195–202.

[9] J. H. Anderson, V. Bud, and U. C. Devi, "An EDF-based restricted-migration scheduling algorithm for multiprocessor soft real-time systems," *Real-Time Systems*, vol. 38, no. 2, pp. 85–131, 2008.

[10] S. K. Lee, "On-line multiprocessor scheduling algorithms for real-time tasks," in *Proceedings of the IEEE Region 10's Ninth Annual International Conference (TENCON)*, vol. 2. IEEE Computer Society, August 1994, pp. 607–611.

[11] R. I. Davis and A. Burns, "Fpzl schedulability analysis," in *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Chicago, IL, USA: IEEE Computer Society, April 2011, pp. 245–256.

[12] 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)*. Pozman, Poland: IEEE Computer Society, June 1994, pp. 162–171.

[13] L. Cucu and J. Goossens, "Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors," in *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, September 2006, pp. 397–405.

[14] J. Goossens and R. Devillers, "The non-optimality of the monotonic priority assignments for hard real-time offset free systems," *Real-Time Systems*, vol. 13, no. 2, pp. 107–126, September 1997.

[15] S. K. Baruah and N. W. Fisher, "Global fixed-priority scheduling of arbitrary-deadline sporadic task systems," in *Proceedings of the 9th International Conference on Distributed Computing and Networking (ICDCN)*, S. Rao, M. Chatterjee, P. Jayanti, C. S. R. Murthy, and S. K. Saha, Eds., vol. 4904/2008. Kolkota, India: Springer, January 2008, pp. 215–226.

[16] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, 2010.

[17] E. Bini and G. C. Buttazzo, "Biasing effects in schedulability measures," in *Proceedings of the 16th Euromicro Conference on Real-time Systems (ECRTS)*. Catania, Sicily, Italy: IEEE Computer Society, June - July 2004, pp. 196–203.

[18] N. W. Fisher, T. P. Baker, and S. K. Baruah, "Algorithms for determining the demand-based load of a sporadic task system," in *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. Sydney, Australia: IEEE Computer Society, August 2006, pp. 135–146.

[19] M. Bertogna, "Evaluation of existing schedulability tests for global EDF," in *Proceedings of the 38th International Conference on Parallel Processing Workshops (ICPPW)*. Vienna, Austria: IEEE Computer Society, September 2009, pp. 11–18, first International Workshop on Real-time Systems on Multicore Platforms: Theory and Practice (XRTS).