Job vs Portioned Partitioning for the Earliest Deadline First Semi-Partitioned Scheduling

Laurent George

University of Paris-Est Creteil, LISSI, 120 rue Paul Armangot, 94400 Vitry sur Seine, France

Pierre Courbin

ECE Paris, LACSC, 37 quai de Grenelle, 75015 Paris, France

Yves Sorel

Yves Sorel, INRIA, AOSTE, Domaine de Voluceau - B.P. 105, 78153 Le Chesnay, France

Abstract

In this paper, we focus on the semi-partitioned scheduling of sporadic tasks with constrained deadlines and identical processors. We study two cases of semi-partitioning: (i) the case where the *Worst Case Execution Time* (WCET) of a job can be portioned, each portion being executed on a dedicated processor, according to a static pattern of migration; (ii) the case where the jobs of a task are released on a processor, 1 time out of p, where p is an integer at most equal to the number of processors, according to a Round-Robin migration pattern. The first approach has been investigated in the state-of-the-art by migrating a job at its local deadline, computed from the deadline of the task it belongs to. We study several local deadline assignment heuristics (fair, based on processor utilization and based on the minimum acceptable local deadline for a job on a processor). In both cases, we propose feasibility conditions for the schedulability of sporadic tasks scheduled using *Earliest Deadline First* (EDF) semi-partitioned scheduling. We show that the load function used for global scheduling to establish the feasibility of sporadic task sets exhibits interesting properties in the semi-partitioning context. We carry out simulations to study the performance of the two approaches in terms of success rate and number of migrations, for platforms composed of four and eight processors. We compare the performance of these semi-partitioned heuristics with the performance of classical partitioned scheduling algorithms and with a global scheduling heuristic which is currently considered to have good performances.

Keywords: Real-Time, Multiprocessor, Semi-partitioned scheduling, portioned scheduling, EDF, deadline assignment, performance evaluation

Email addresses: lgeorge@ieee.org (Laurent George), courbin@ece.fr (Pierre Courbin), Yves.Sorel@inria.fr (Yves Sorel)

1. Introduction

In this paper, we consider the problem of real-time identical multiprocessor scheduling of sporadic tasks in the constrained deadline case ([1]), with no parallelism in the execution. Multiprocessor scheduling is an active area of research that has mostly been studied using partitioned scheduling approaches. In the partitioned scheduling case, tasks are assigned to the processors according to a partitioning heuristic and cannot migrate. A classical uniprocessor scheduling feasibility condition is then used to decide on the schedulability of the tasks.
Partitioned scheduling is attractive as it does not lead to job migration costs that can influence the schedulability of the system. However, it has been shown that in some cases, sufficient feasibility conditions can only ensure the feasibility of a system with a system utilization of less than 50% ([2],[3]). This is an indication of the pessimism of partitioned scheduling.
Another approach called *global scheduling* is considered to have theoretically better performances compared to partitioning approaches. With global scheduling, jobs are allowed to migrate and processor utilization can reach 100% ([4],[5],[6]). Migration costs particularly depend on cache-related migration delays and on the interconnect length.

A recent paper [7] has shown, with an experimental study done on a 24-core Intel Uniform Memory Architecture (UMA) machine, that cache-related preemption and migration delays were comparable for a system under load. Furthermore, recent advances in 3D multicore technology [8] reduce the interconnect lengths, thus reducing the migration delays. In addition, migration costs can also be reduced by optimizing migration policies to amortize the cost of a task migration [9](e.g. lazy state transfer, precopying, low level migration). All this would appear to make scheduling with migration an attractive solution. However, migration costs in current feasibility conditions are either not taken into account or, if they are, only by inflating task execution times, which might not be accurate and may lead to CPU over-provisioning.

Optimal strategies have been proposed for periodic real-time tasks: Pfair ([4],[5]), where each task is divided into quantum-size pieces having pseudo deadlines, and LLREF ([6]), with T-L plane abstraction where the scheduling is done to bound the number of preemptions. These strategies, although optimal, can lead to a large number of migrations, thus leaving their applicability to real-time systems uncertain. An active area of research aims to tackle the problem of reducing the number of job migrations, to reduce the impact of migration cost on the feasibility conditions (see section 3.2). Nevertheless, only sufficient feasibility conditions have been proposed for global scheduling. These sufficient feasibility conditions are in the current state-of-the-art more pessimistic than the feasibility conditions obtained for partitioned scheduling ([10]).

More recently, a semi-partitioning approach has been proposed as a possible approach. This approach can be seen as an intermediate solution between partitioned and global scheduling approaches. In classical semi-partitioned scheduling, the goal is to control the number of job migrations in order to reduce run-time overheads. The basic idea with semi-partitioned scheduling is to execute tasks according to a static job migration pattern. Most results propose heuristics that first try to assign, as much as possible, tasks to a single processor according to a particular partitioned scheduling heuristic. The jobs of the

tasks that cannot be assigned to a single processor are then allowed to migrate between a fixed set of particular processors. The state-of-the-art proposes several approaches:

1. The approach that splits the WCET of a task into several portions, each portion being successively executed on a dedicated processor [3]. This case is referred to as *job portion migration*. The state-of-the-art has considered the optimistic case where no constraint is imposed on the possibility of splitting WCETs. We focus on the solution that consists of migrating a job on a processor at a local deadline associated to a portion of WCET. The local deadline is computed from the deadline of the task. We study in section 5.1.1 several local deadline assignment heuristics.
2. The approach that defines a migration pattern of successive jobs, referred to by Funk & al. in [11] as the *restricted migration* case. A job is executed on only one processor but different jobs of the same task can be executed on different processors. This migration pattern introduces fewer overheads than the *job portion migration* approach as migrations are only done at job boundaries (at most one migration per job).

In this paper, we focus on both approaches, considering Earliest Deadline First (EDF) scheduling [12].

Our contributions:

For the first approach, we show how to generalize the approaches given in the state-of-the-art of semi-partitioned scheduling to the general case of schedulers applying jitter cancellation before migrating a job. Jitter cancellation is a classical approach used in networked systems to reshape the arrival pattern of incoming flows. In networked systems, jitter control is done according to the minimum and maximum response times of incoming flows at destination nodes. In our context, this is done on the processor that has to migrate a job. The basic idea is to postpone the migration of a job on a processor as long as it has not reached its maximum response time. With jitter cancellation, we have the interesting property that the release time pattern of all the jobs of a task is identical for all the processors executing the task. A classical uniprocessor feasibility condition can be applied independently on each processor that executes a task.
Jitter cancellation is obtained by migrating jobs at local deadlines, computed from the deadlines of their task. The benefit of this solution is that adding a new task to a processor only impacts on the feasibility of the task assigned to the processor (once assigned, the local deadline of a task cannot change). We just need to check that the local deadlines of all the tasks assigned to this processor are met. The main drawback of this solution is the possible rejection of a feasible task configuration that meets the global deadline but not all the local deadlines. For the first approach, our main contributions are as follows:

- We recall local deadline assignment heuristics that have been considered in the state-of-the-art in section 5.1.1 and study their performance using simulations in section 6.

- We revisit the heuristic of Kato & al. (see [3]), where the local deadline of a job is fair (equal to the deadline of the task divided by the number of processors visited

by a job). To our knowledge, this heuristic has the best performance in terms of the success ratio. We extend this solution as follows:

  - By proposing a sensitivity analysis on the WCETs to decide on the duration of a portion of a subtask assigned to a processor. We establish new results on sensitivity analysis in section 4 to compute the maximum feasible duration of a portion of the WCET with EDF scheduling.
  - By proposing several local deadline assignment heuristics from the state-of-the-art and a local deadline assignment heuristic denoted as EDF-MLD-Dmin, based on the minimum acceptable local deadlines for a job on the processors executing it. In section 6, we show by simulations, the performance of these local deadline assignment heuristics. The solution based on local deadlines computed from minimum acceptable deadlines in some cases outperforms existing local deadline assignment heuristics in terms of the success ratio (see section 6).

For the second approach, the jobs of a task are migrated before releasing a new job according to a static job migration pattern. Our main contribution for approach 2 is to propose a Round Robin Job Migration (EDF-RRJM) heuristic (see subsection 5.2). We establish a generic feasibility condition for EDF-RRJM scheduling, based on the *load* function. This feasibility condition is then applied on each processor.

The remainder of this paper is organized as follows: In section 2, we introduce the task model and the notations used. Section 3 reports to to the state-of-the-art for partitioned, global and semi-partitioned scheduling algorithms. We provide a table comparing the characteristics of semi-partitioned heuristics. Section 4 revisits the feasibility conditions for EDF scheduling, showing that the *load* function used in global scheduling can also be used to establish new feasibility conditions for semi-partitioned scheduling. We then revisit the problem of computing the minimum acceptable deadline for a task, providing corrections to some problems we have identified with the original result. Section 5 focuses on portioned semi-partitioned and job scheduling with Round Robin migrations. For portioned scheduling, we show that this problem can be solved by migrating jobs at local deadlines, based on the deadlines of the tasks. We give a generic feasibility condition for EDF valid for any local deadline assignment heuristic. Finally, we describe a portioned scheduling algorithm using local deadlines computed from the minimum local deadline of the tasks. We then present our second approach based on Round Robin job migration and we establish the associated feasibility conditions for EDF scheduling. For a task that cannot be assigned to a single processor, we execute a job 1 time out of $p \leq m$ on a set of p processors, where m is the total number of processors. Section 6 is devoted to simulation results. We use simulations to study the performance of EDF feasibility conditions for both approaches. We consider the following scheduling algorithms:

- Four portioned semi-partitioning heuristics based on approach 1, with local deadlines:
  - Computed according to the solution proposed in Kato $\&$ al. ([3]), with migrations at local fair deadlines.

  - One based on a variant of Kato & al with fair deadlines and fair portions
  - One based on local deadlines proportional to processor utilizations.
  - One based on local deadlines as a function of the minimum acceptable deadline on each processor (EDF-MLD-Dmin).
- One based on Round Robin Job Migrations heuristic (EDF-RRJM).
- A well known global EDF scheduling [13].

We use simulations to compare the algorithms in terms of (i) success rate for accepting tasks on a set of $m = 4$ and $m = 8$ processors in the case of arbitrary loads for First-Fit (FF) and Worst-Fit (WF) bin packing algorithms and (ii) in terms of density of job migrations. Finally we summarize our conclusions in section 7.

2. System Model

In this paper, we consider the problem of scheduling a sporadic task set on a multiprocessor system. We suppose a set of m identical processors, executing jobs at the same processor speed. A sporadic task set $\tau_{(C,T,D)}$ is a task set composed of n sporadic tasks, where C, T and D respectively denote the sets of WCETs, Periods and Deadlines of the tasks. We also use the notation τ w.r.t. $\tau_{(C,T,D)}$ to simplify the notations when possible. Each sporadic task $\tau_i(C_i, T_i, D_i)$ is defined by:

- C_i: its worst-case execution time (WCET), obtained on any processor;
- T_i: its minimum inter-arrival time (also called, by extension, the period);
- D_i: its relative deadline.

Each task τ_i generates an infinite set of jobs with a minimum inter-arrival time T_i. For a job of τ_i released at time t_i, $t_i + D_i$ is its absolute deadline.

In the state-of-the-art [14], the case where respectively $\forall \tau_i, D_i = T_i$ (respectively $\forall \tau_i, D_i \leq T_i$) is referred to as the *implicit deadline* (respectively *constrained deadline*) case. The case where no specific relation between periods and deadlines is valid for all the tasks is referred to as the *arbitrary deadline* case. In the following, we assume constrained deadlines and the following definitions:

- $\lambda_i = \frac{C_i}{min(T_i, D_i)}$ is the density of $\tau_i(C_i, T_i, D_i)$.
- $h(t) = \sum_{i=1}^{n} h_i(t)$, denotes the processor Demand Bound Function (DBF, [12]) where $h_i(t) = Max(0, 1 + \lfloor \frac{t-D_i}{T_i} \rfloor)C_i$. $h(t)$ is the workload resulting from the execution of all the jobs that have their absolute deadlines in the time interval $[0, t]$, in the synchronous scenario (the first jobs of the tasks are released at time 0).
- $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ is the processor utilization resulting from $\tau_{(C,T,D)}$.
- $P = LCM(T_1, \ldots, T_n)$ is the Least Common Multiple of the periods of the tasks, also called the hyperperiod.

- The scheduling considered is Earliest Deadline First (EDF) preemptive scheduling. EDF schedules jobs that have shorter absolute deadlines first. Ties are broken arbitrarily.
- $\pi = \{\pi_1, \ldots, \pi_m\}$ is the set of m identical processors in charge of executing the tasks.
- $\tau^{(k)}_{(C,T,D)}$ denotes the subset of tasks in $\tau_{(C,T,D)}$ executed on processor π_k. We also use the notation $\tau^{(k)}$ to simplify the notations when possible.
- $C_i^{(k)}$ denotes the duration of a task (or subtask) τ_i assigned to processor π_k.

3. Related Work

In the uniprocessor case, EDF scheduling, has been proved to be optimal for scheduling sporadic tasks. Optimality is defined as the property that any schedulable task set is always schedulable with EDF (all the jobs of tasks meet their deadlines). Optimality of EDF is no longer guaranteed in the multiprocessor case. Three main approaches have been considered for scheduling sporadic tasks on a multiprocessor system: the partitioning approach (see subsection 3.1), the global scheduling approach (see subsection 3.2) and the semi-partitioning approach (see subsection 3.3).

3.1. Partitioned scheduling

With the partitioning approach, we need to find a partitioning heuristic to assign tasks to processors and then to use a uniprocessor feasibility condition on each processor to decide on the schedulability of the task assigned to it.

The problem of finding a feasible partitioning is a bin packing problem known to be NP hard in the strong sense [15]. The state-of-the-art therefore focuses on different partitioning heuristics. These include First-Fit, Next-Fit, Best-Fit and Worst-Fit [2], [16]. First-Fit partitioning heuristics have received more attention.

- First-Fit (FF): tasks are allocated sequentially, one by one to the first processor it fits into (according to the utilization or the DBF criterion). The process always starts from processor π_1 up to processor π_m.
- Next-Fit (NF): tasks are allocated sequentially, one by one to the first processor it fits into (according to the utilization or the DBF criterion). The process always starts from the last processor where a task has been assigned (the first processor for the first task).
- Best-Fit (BF): tasks are allocated sequentially but a task is assigned to the processor it fits best so that it will minimize the remaining processor capacity (in the sense of the utilization criterion or DBF criterion).
- Worst-Fit (WF): the same as BF except that the goal is to maximize the remaining processor utilization.

Notice that the goal of BF is to minimize the number of processors by maximizing the load on each processor. WF can be interesting to spread the tasks over as many processors as possible to share the load, for fault-tolerance issues, for example to tolerate WCET overruns ([17]). A variant of these allocation algorithms is first to sort the set of tasks to be assigned in decreasing values of task density or processor utilization, leading to FF Decreasing (FFD), BF Decreasing (BFD) or WF Decreasing (WFD) variants.
Examples of First-Fit partitioning are:

- Density-based partitioning. For each task, a density computed from the task parameters is considered for the partitioning (see [18] for an exhaustive list of density-based partitioning heuristics).
- Partitioning based on the demand bound function (DBF) approximation ([14], [1]).

Among these density-based partitioning heuristics, we have, for any task $\tau_i(C_i, T_i, D_i)$, the utilization parameter $\frac{C_i}{T_i}$ or the density parameter $\lambda_i = \frac{C_i}{min(D_i,T_i)}$ ([14], [1]). On a given processor π_k, the following condition is used to assign tasks in τ to processor π_k:

- $\diamond$ $\sum_{\tau_i(C_i,T_i,D_i)\in\pi_k} C_i/T_i \leq 1$ which is only a necessary feasibility condition for τ with EDF on processor π_k. For constrained deadlines, a second step is necessary to check whether the deadlines can be met while on π_k.
- $\diamond$ $\sum_{\tau_i(C_i,T_i,D_i)\in\pi_k} C_i/min(D_i,T_i) \leq 1$ is sufficient ([19]) to check the feasibility of a task set on π_k. The density partitioning will lead to a feasible partitioning on $m \geq 2$ processors if the following condition is satisfied ([20]):

$$\sum_{i=1}^{n} \lambda_i \leq \begin{cases} m-(m-1)\lambda_{max}(\tau), \text{if} \lambda_{max}(\tau) \leq \frac{1}{2} \\ \frac{m}{2} + \lambda_{max}(\tau), \text{if} \lambda_{max}(\tau) \geq \frac{1}{2} \end{cases} \quad (1)$$

where, $\lambda_{max}(\tau) = max_{i=1\ldots n}\lambda_i$

As concerns partitioning based on a DBF approximation, [1] proposes for any task $\tau_i(C_i, T_i, D_i)$, the approximation of $h_i(t)$, denoted as $h_i^*(t)$, where $h_i^*(t) = C_i + U_i \times (t - D_i)$ for any time $t \geq D_i$ and 0 otherwise. The DBF partitioning will assign task $\tau_i(C_i, T_i, D_i)$ to a processor π_k if $\tau_i(C_i, T_i, D_i)$ satisfies the following condition, with already assigned tasks on π_k ([20]):

$$\begin{cases} (D_i - \sum_{\tau_j(C_j,T_j,D_j)\in\pi_k} h_i^*(D_i)) \geq C_i \\ (1 - \sum_{\tau_j(C_j,T_j,D_j)\in\pi_k} U_j) \geq U_i \end{cases} \quad (2)$$

George & al. propose in [16] a Worst-Fit Decreasing heuristic based on the load(τ) function. The load(τ) function is the cumulative execution requirement generated by jobs of the tasks in τ on any time interval divided by the length of the interval and is defined as ([21]):

$$load(\tau) = Sup_{t\in\mathbb{R}^{+*}} \left\{ \frac{h(t)}{t} \right\} \quad (3)$$

The goal of the heuristic given by [16] is to maximize the remaining processor utilization characterized by the function $1 - load(\tau^{(k)})$ on each processor π_k.

Finally, if a sporadic task set is schedulable with the partitioning approach on m identical processors then it can be scheduled with EDF with the DBF partitioning for any task $\tau_i(C_i, T_i, D_i)$, with processors having a speed [14]:

- $\diamond$ $(4 - \frac{2}{m})$ times as fast as the original processor speed for arbitrary deadlines.
- $\diamond$ $(3 - \frac{1}{m})$ times as fast as the original processor speed for constrained deadlines.

3.2. Global scheduling

A recent paper [7] has shown through some experiments that cache migration delays can be equivalent to preemption delays for a system under load. The evolution of current 3D architectures [8] also tends to reduce migration-related penalties. Thus, it would be interesting to compare the schedulability of partitioned and semi-partitioned algorithms with some of the main results for the schedulability of global EDF:

- Goossens & al. [22] prove a utilization-based schedulability test called GBF.
- Baker [23, 24] offers a different approach based on the analysis of the workload. This test is similar to GBF in implicit deadline systems but incomparable for constrained deadline systems.
- Baruah [25] proposes a parallel condition derived from the computation of the demand bound function.
- Baker and Baruah [26] base their schedulability test on the computation of the *load* function. Previous sufficient schedulability tests related to the *load* function have been presented but this test is shown to surpass them.
- Bertogna & al. [27] present an iterative approach based on the slack of each task. This information is used to estimate the interfering workload in a scheduling window. Bertogna has named this test BCL.
- Bertogna & al. [13] introduce RTA which is a schedulability test based on an iterative estimation of the worst case response time of each task.
- Baruah & al. [28] focus on the demand bound function to derive a sufficient schedulability test. This test has the smallest possible processor speedup factor of $(2 - \frac{1}{m})$ for EDF.

In [10], Bertogna compares the main existing results in this area. All these conditions are evaluated according to the number of task sets that are detected to be schedulable. Since these schedulability tests are incomparable in terms of task sets detected schedulable, Bertogna proposes the algorithm COMP based on the sequence of the best previous techniques. According to this study, COMP and RTA appear to detect the largest number of schedulable task sets. Since the improvement given by COMP seems limited compared

to the amount of additional computation required, we chose to implement RTA in this study.
The procedure of RTA is implemented according to that presented in [10]. They show how to compute iteratively R_k^{ub}, an estimated worst case response time of each task τ_k.

The iteration starts with $R_k^{ub} = C_k$ and ends if either the recursive equation has converged or $R_k^{ub} > D_k$. S_i^{lb} represents the slack of the task and is initialized to 0. If the operation fails, τ_k is marked as "potentially not schedulable", otherwise we assign $S_i^{lb} = D_k - R_k^{ub}$.
The equation of R_k^{ub} is as follows :

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} \min\left(\mathfrak{W}_i(R_k^{ub}), \mathfrak{I}_k^i, R_k^{ub} - C_k + 1\right) \right\rfloor \tag{4}$$

$$\text{with : } \mathfrak{W}_i(t) = \left\lfloor \frac{t + D_i - C_i - S_i^{lb}}{T_i} \right\rfloor C_i + \min\left(C_i, (t + D_i - C_i - S_i^{lb}) mod T_i\right) \tag{5}$$

$$\mathfrak{I}_k^i = \left\lfloor \frac{D_k}{T_i} \right\rfloor C_i + \min\left(C_i, \max(D_k mod T_i - S_i^{lb}, 0)\right) \tag{6}$$

After having computed all R_k^{ub}, if no task has been marked as "potentially not schedulable", we declare the task set to be schedulable. Otherwise, a new iteration is computed. We stop if the task set is declared as schedulable or if, in the last iteration, no slack has been updated. In this case, the task set is declared as being not schedulable.

3.3. Semi-partitioned scheduling

The concept of semi-partitioned scheduling was introduced by Anderson & al in [29] where the authors define two classes of tasks : those assigned to only one processor, and those assigned to two different processors. Tasks assigned to two processors are called *migrating* tasks while those assigned to only one processor are called ***fixed*** tasks.

Anderson & al also define the degree of migration allowed by an algorithm:

1. No migration (i.e., task partitioning)
2. Migration allowed, but only at job boundaries (i.e., migration at the job level). A job is executed on one processor but successive jobs of a task can be executed on different processors. In this paper, we consider *Round Robin job migrations*.
3. Job portion migration (i.e., jobs are also allowed to migrate during their execution at defined times).

EDF-fm [29] belongs to the second category. It splits jobs between two processors allocating *r* jobs over *s* to a processor with the index *p*, and the other jobs (*s-r* over *s*) to a processor with the index *p+1*. The number of migrations is reduced and the total utilization of this task can be adapted on each processor. However, EDF-fm is best suited to soft real-time systems since it cannot guarantee the deadlines of fixed tasks.

In terms of migrations, the following algorithms are classified in the third category (job portion migration). They split tasks according to their WCET between two or more processors. Parts of the migratory job are executed on separate processors, but simultaneity

of execution is not allowed.

Anderson & al. [29] lay the foundations for the assignment of tasks on processors. The principle is to fill each processor sequentially. If the remaining capacity of a processor with index *p* is not large enough to receive the entire task, this task is split into two parts. The first part is assigned to fill processor *p* and the second part is assigned to processor *p+1*. Thus, there are at most two migratory tasks on each processor and *m-1* migratory tasks in the whole system. This technique is similar to a *Next-Fit bin-packing* heuristic with *task splitting*. All the following algorithms up to EDHS[30] use this assignment.

- EKG[31] offers a complex but optimal solution to this problem. According to a parameter *K* which defines the size of each group of processors accepting migratory tasks, EKG is able to adapt the utilization bound and the number of preemptions. Since a migration only happens when a task is preempted, reducing the number of preemptions tends to reduce the number of migrations. Although when $K = m$, EKG is optimal with a utilization bound of 100%, it incurs more preemptions.

- EDDHP (originally named Ehd2-SIP)[32] reduces the number of preemptions and increases the success ratio with regard to partitioned algorithms. EDDHP is outperformed by EKG in terms of its success ratio but is more convenient to implement and to use in practical cases.

- Kato & al introduce the notion of *portion* and named their algorithms *portioned scheduling*. In [32], the utilization is shared on two processors. Furthermore, the authors propose an optimization that will be important subsequently. They may find a task set unschedulable according to their algorithm but schedulable with a simple partitioning algorithm such as EDF-FF. It proves that their splitting method may degrade schedulability compared to some nonsplitting methods. Thus, they optimize their algorithm to deal with this case.

- EDDP [33] improves the schedulability of EDDHP by introducing some mechanisms of EKG. Indeed, these algorithms distinguish two types of tasks based on their utilization: light tasks and heavy tasks. EDDP is still easier to implement than EKG and guarantees a new utilization bound of 65% with fewer migrations.

- RMDP [34] is a static priority version of EDDHP. Kato & al. claim that a static priority scheduling is still widely used and it does not suffer from the domino-effect problem or the disadvantage of varying jitter in periodic execution.

- EDHS [30] suggest fundamentally changing the assignment of tasks on processors. For all previous algorithms, except for the optimization of EDDHP, if some task causes the total utilization of a processor to exceed its utilization bound, the WCET of this task is always split into two portions. For EDHS, a simple partitioning is performed before splitting the WCET of a task. If the partitioned scheduling fails, the remaining WCET portions are shared on two or more processors. Each part of the task is defined in order to fill a processor. Kato & al. chose to attribute at most one migrating task to each processor. A task always migrates in the same way, between the same processors and at the same time of their execution. Here, the notion of *semi-partitioned* scheduling takes its full meaning.

- DM-PM [35] is a static priority version of EDHS. If tasks are sorted by decreasing deadlines before assignment, migratory tasks naturally have a higher priority than fixed tasks. The scheduling of migratory tasks is thus easier.

- EDF-WM [3] tries to adapt the simplification introduced in DM-PM to dynamic priority scheduling. Thus, a task is split according to its WCET but its deadline is also portioned into local deadlines used on each processor executing the task. This defines a window during which a subtask should be executed. The local deadline of a task $\tau_i(C_i, T_i, D_i)$ is equal to D_i/s (fair local deadline) where s is the number of processors executing the task. WCETs are chosen to fill the processor w.r.t. to the fair local deadline. Schedulability analysis and complexity of the scheduler are improved with this technique. The implementation is also easier if we consider subtasks as independent tasks with a delayed release time.

- SPA2 [36] assigns tasks according to a Worst-Fit heuristic. The tasks that cannot be assigned to a single processor are split into subtasks such that the sum of the WCETs of all the subtasks is equal to the WCET of the task. A subtask released at time t on a processor π_k is migrated at time t+R where R is the worst case response time of the subtask on π_k. The authors show that the utilization bound obtained with SPA2 is equal to $n(2^{\frac{1}{n}} - 1)$, the Liu $\&$ Layland's bound defined in [37], that tends towards 69.3%, where n is the number of tasks.

- In [38], the authors introduce the EDF-SS($DMIN/\delta$) algorithm. The basic idea of this algorithm is to split tasks that cannot be scheduled on only one processor, between two processors. The WCETs of these tasks are divided into slots of length equal to $DMIN/\delta$ where $DMIN$ is the minimum of all deadlines and periods and $\delta > 1$ is an integer parameter that is configurable. The smaller the value, and the smaller the slot size, the more migrations. The slots reserved for a task on any two different processors are synchronized in time. Tasks that are split have a higher priority than tasks executed on a single processor. This approach was first considered in the case of implicit deadlines in [39].

- In [40], the authors introduce the PDMS_HPTS_DS algorithm based on Partitioned Deadline-Monotonic Scheduling (PDMS) with the Highest Priority Task Split (HPTS) heuristic. With this approach, the task having the highest priority on a processor that cannot be executed on a single processor is split between two processors. Tasks are allocated in the Decreasing order of Size (density in our paper). The authors assign local deadlines for a task τ_i (highest priority) equal to D_i on the first processor executing τ_i and $D_i - C_i^{(first)}$ on the second processor, where $C_i^{(first)}$ is the WCET of τ_i on the first processor, also equal to its worst case response time. They show that the PDMS_HPTS_DS achieves a utilization bound of 65% (compared to 50% [2],[3]).

In table 1, we summarize the properties of the semi-partitioned scheduling algorithms described above.
We can conclude from this state-of-the-art of semi-partitioned scheduling that the tendency is to find an algorithm able to schedule more task sets than both partitioned scheduling with fewer migrations and global scheduling. The complexity of implementation is

Table 1: Some existing algorithms based on semi-partitioned scheduling

Name	Ref.	System properties	Priority of migratory tasks	Scheduling of non-migratory tasks
EDF-fm	[29]	Implicit Deadlines Sporadic/Periodic Identical processors	Highest/Fixed priority	Managed with EDF
EKG	[31]	Implicit Deadlines Sporadic/Periodic Identical processors	Highest/Fixed priority	Managed with EDF
EDDHP	[32]	Implicit Deadlines Periodic	Highest/Fixed priority	Managed with EDF
EDDP	[33]	Implicit Deadlines Periodic	Managed with EDF and specific priorities between portions to ensure non-simultaneity of execution	Managed with EDF
RMDP	[34]	Implicit Deadlines Periodic	Highest/Fixed priority	Managed with RM
EDHS	[30]	Implicit Deadlines Sporadic/Periodic	Highest/Fixed priority	Managed with EDF
DM-PM	[35]	Constrained Deadlines Sporadic/Periodic Identical processors	Highest/Fixed priority	Managed with DM
EDF-WM	[3]	Arbitrary Deadlines Sporadic/Periodic Identical processors	Managed with EDF. WCETs are split to fill processor utilization. Migration at local fair deadlines	Managed with EDF
SPA2	[36]	Implicit Deadlines Sporadic/Periodic Identical processors	Highest/Fixed priority Migration at local Worst Case Response Time	Managed with RM
EDF-SS($DMIN/\delta$)	[38], [39]	Arbitrary Deadlines Sporadic/Periodic	Highest/Fixed priority. WCETs are split into reserved slots of length $DMIN/\delta$	Managed with EDF
PDMS_HPTS_DS	[40]	Constrained Deadlines Sporadic/Periodic	Highest/Fixed priority. WCETs are split into two portions Local deadlines for each portions	Managed with DM

also a point to consider. EDF-fm is based on migrations at job boundaries which leads to a simple implementation but the version proposed is only designed for soft real-time scheduling. Other algorithms presented in this study focus on job portion migration and split tasks into subtasks, of WCETs based on the WCETs of the tasks. This leads to an optimal algorithm (EKG) but this solution is quite difficult to implement. With suboptimal algorithms, Kato & al. were able to achieve simpler algorithms with reasonable utilization bound and fewer migrations. We initially study this problem (see section 5.1), when the migration of a job occurs at its local deadline.

However those approaches require using an operating system that keeps track of job cpu consumption in order to migrate a job when it has been executed. Many operating systems offer execution overrun timers to specify that a job has been executed for a given duration (e.g. [41] in AUTOSAR OS, [42] in Real-Time Specification for Java: RTSJ). Nevertheless, the migration time is not necessarily identical to the time when an execution overrun occurs. This might introduce time overhead in the management of such timers to adapt them to task migration. Moreover, a migration during the execution of a job requires transferring the execution context between processors, which can prove costly on a multiprocessor system.

In this paper, we also consider another approach, based on Round Robin job migrations (EDF-RRJM), presented in section 5.2. It is based on migrations at job boundaries.

4. Revisiting uniprocessor feasibility conditions

4.1. Revisiting the load($\tau_{(C,T,D)}$) function

Let $\tau_{(C,T,D)}$ be a task set of n sporadic tasks where $C = (C_1, \ldots, C_n)$, $T = (T_1, \ldots, T_n)$ and $D = (D_1, \ldots, D_n)$ are respectively the set of WCETs, periods and deadlines of the tasks. A task $\tau_i(C_i, T_i, D_i)$ is defined by the i_{th} elements of the three sets C,T and D.
We recall that the load($\tau_{(C,T,D)}$) is the cumulative execution requirement generated by jobs of the tasks in τ on any time interval divided by the length of the interval. The load($\tau_{(C,T,D)}$) function is defined as [21]:

$$load(\tau_{(C,T,D)}) = Sup_{t \in \mathbb{R}^{+*}} \left\{ \frac{h(t)}{t} \right\} \tag{7}$$

The load($\tau_{(C,T,D)}$) function has been considered as an interesting function to provide a sufficient feasibility condition for global EDF scheduling ([43]). In the uniprocessor context, the load($\tau_{(C,T,D)}$) function provides a necessary and sufficient feasibility condition for EDF on a uniprocessor: $load(\tau_{(C,T,D)}) \leq 1$. Fisher & al, in [21], show how to compute the load($\tau_{(C,T,D)}$) function, leading to theorem 1 (we recall that P is the least common multiple of the periods):

Theorem 1. *[21] Let $\tau_{(C,T,D)}$ be a sporadic task set.*

$load(\tau_{(C,T,D)}) = Sup_{t \in \mathbb{R}^{+*}} \left\{ \frac{h(t)}{t} \right\} =$

$Max \left\{ U, Sup_{t \in \mathcal{S}} \left\{ \frac{h(t)}{t} \right\} \right\} \leq 1$, *where* $\mathcal{S} = \bigcup_{j=1}^{n} \left\{ D_j + k_j T_j,\ 0 \leq k_j \leq \left\lceil \frac{P - D_j}{T_j} \right\rceil - 1 \right\}$.

In [44], George & al. show that this expression can be used to characterize the space of feasible WCETs. Considering the WCETs in $X = (x_1, \ldots, x_n)$ as variables and D, T as constants, the $load(\tau_{(X,T,D)}) \leq 1$ condition defines a set of $s + 1$ constraints, where the first s constraints are derived from the set of elements in $\mathcal{S}$ where s is the number of elements in $\mathcal{S}$ and the $(s+1)^{th}$ constraint is derived from the load utilization ($U \leq 1$).

They show how to prune the set $\mathcal{S}$, for the computation of load($\tau_{(X,T,D)}$), to extract the subset of elements in $\mathcal{S}$ representing the most constrained times where $Sup_{t \in \mathbb{R}^{+*}} \left\{ \frac{h(t)}{t} \right\}$ can be obtained.
For any time $t_i \in \mathcal{S}$, they formalize as a linear programming problem the question of determining whether a time t_i is relevant in $\mathcal{S}$ or if it could be ignored. For each time t_i the goal is to maximize the objective function $h(t_i)$ taking into account the $s - 1$ constraints, $h(t_k) = \sum_{j=1}^{n} Max(0, 1 + \lfloor \frac{t_k - D_j}{T_j} \rfloor)\, x_j \leq t_k$, $1 \leq k \leq s$, $k \neq i$, where x_j is the the variable representing the WCETs of $\tau_j(x_j, T_j, D_j)$. These $s-1$ constraints are constraints imposed on the WCETs of the tasks without considering the constraint associated to time t_i.
The problem to be solved can be characterized with a linear programming approach, and is more formally defined as follows:

Linear Programming Problem: LP 1.

Maximize $h(t_i)$
With $x_1 \geq 0, \ldots, x_n \geq 0$ positive real variables
Under the constraints:
$\bigcup_{k=1,k\neq i}^{s} \{h(t_k) \leq t_k\}$

[44] shows that the space of feasible WCETs is convex. They propose using the simplex algorithm to solve LP1. If for time t_i, $h(t_i) \leq t_i$ when the $s-1$ constraints on the WCETs $h(t_k) \leq t_k$, $1 \leq k \leq s$, $k \neq i$, are imposed, then adding the constraint $h(t_i) \leq t_i$ for time t_i will add more constraints on the possible values of WCETs $(x_1, \ldots, x_n)$. This can only reduce or leave unchanged the goal function $h(t_i)$. Hence t_i is not significant for characterizing the space of feasible WCETs and can then be removed from $\mathcal{S}$. Otherwise, time t_i should be kept in $\mathcal{S}$.

Performance of the linear programming approach LP1:
We now study the performance of the simplex for pruning the elements in S, in the case of constrained deadlines ($\forall i \in [1, n], D_i \leq T_i$) using an exhaustive example with more than 3500 constraints in the set S for all the task sets considered. Notice that the number of constraints in S depends more on the value of the periods than on the number of tasks (the number of constraints can be small even for a high number of tasks).
In order to evaluate this impact of the simplex, applied to problem LP1, on the reduction of the elements in S, we produce 100,000 systems of three tasks. For each system, we proceed as follows:

- The period of each task is uniformly chosen from [1,100]
- The deadline of any task $\tau_i(C_i, T_i, D_i)$ is $D_i = \alpha T_i$. α is discretized in the intervals $[0, 0.8]$ and $[0.8, 1]$ with a granularity of respectively 0.1 and 0.025.

We focus on the influence of α on the pruning of $\mathcal{S}$ after executing the simplex on the linear programming problem LP1.
Figure 1 shows the results of our analysis. The number of elements in $\mathcal{S}$ is represented by the solid line and associated to the left axis as a function of α. The dotted line refers to the number of elements obtained after the simplex is applied to the linear programming problem LP1 and must be read according to the right axis, as a function of α.
We notice that the number of elements which curb the C-Space inch-up in [0.1, 0.6] then plunge downwards when α tends toward 1. If $\alpha = 1$, we have the special case of implicit deadlines where the only constraint is the processor utilization constraint: $U = \sum_{i=1}^{n} \frac{x_i}{T_i} \leq 1$.
In all cases, we found that the number of constraints before and after pruning the set $\mathcal{S}$ is respectively higher than 3570 and less than 12. This reduction of elements is valid in our example for any processor utilization at most equal to 1. For a load less than 0.6, the average number of constraints in $\mathcal{S}$ after pruning the elements in $\mathcal{S}$ is at most equal to 4. This confirms that the simplex can be very effective to reduce the number of elements characterizing the space of feasible WCETs. We use this property in the semi-partitioned

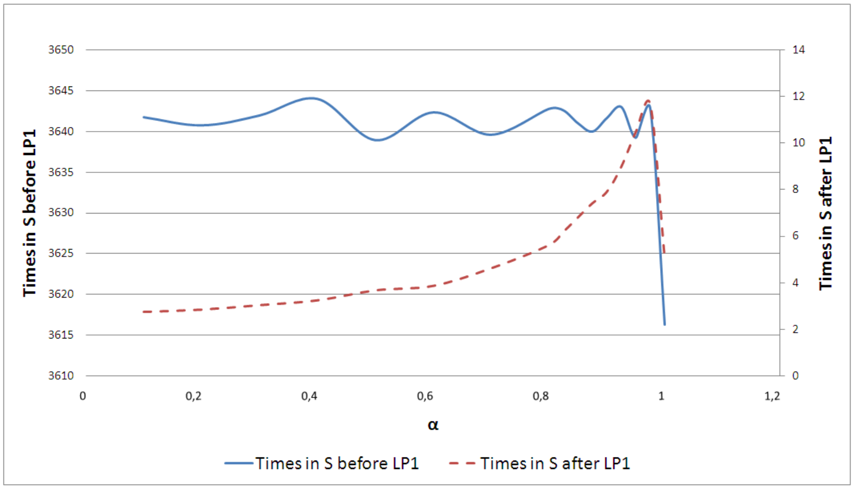

Figure 1: Reduction of elements in $\mathcal{S}$ with LP1

section for the portioned partitioning approach (see section 5)

The load($\tau_{(C,T,D)}$) function has the following properties (see [16]) that we use in section 5 for the problem of semi-partitioned scheduling.
The following property shows that once we have computed the load($\tau_{(C,T,D)}$) for a task set τ, it is straightforward to compute the load of a task set where all the tasks in $\tau_{(C,T,D)}$ have their WCETs multiplied by a real number $\alpha \geq 0$).

Property 1. *Let* $\tau_{(C,T,D)}$ *be a task set.* $load(\tau_{(\alpha C,T,D)}) = \alpha \times load(\tau_{(C,T,D)})$.

The "Sup" of $\alpha h(t)/t$ is equal to α times the "Sup" of $h(t)/t$.

The following property shows that the load of the union of two task sets $\tau_{(C,T,D)}$ and $\beta_{(C',T',D')}$ is at most equal to the sum of the load function of each task set.

Property 2. *Let* $\tau_{(C,T,D)}$ *and* $\beta_{(C',T',D')}$ *be two task sets.*
$load(\tau_{(C,T,D)} \cup \beta_{(C',T',D')}) \leq load(\tau_{(C,T,D)}) + load(\beta_{(C',T',D')})$

The "Sup" of a "Sum" is at most equal to the "Sum" of the "Sups".

Property 3. *Let* $\tau_{(C,T,D)}$ *be a task set. The load corresponding to a transformed task set where all the tasks in* τ *have their period multiplied by* s *is equal to the load corresponding to a transformed task set where all the tasks in* τ *have their WCaat.473(t)-3.05095(-3.087712 3.96 Td [(0)3.*

4.1.1. Example using the linear programing approach to compute the load function
Let us consider the following example $\tau_{(X,T,D)} = \{\tau_1(x_1, T_1, D_1), \tau_2(x_2, T_2, D_2), \tau_3(x_3, T_3, D_3)\}$ of three sporadic tasks given in [44], where, for any task $\tau_i(x_i, T_i, D_i)$, T_i and D_i are fixed, and $x_i \in \mathbb{R}^+$, the WCET of task $\tau_i(x_i, T_i, D_i)$, is a variable.

- $\tau_1(x_1, T_1, D_1) = \tau_1(x_1, 7, 5)$;
- $\tau_2(x_2, T_2, D_2) = \tau_2(x_2, 11, 7)$;
- $\tau_3(x_3, T_3, D_3) = \tau_3(x_3, 13, 10)$.

In this example, we have: $D_{min} = 5$ and $P = 1001$. From Theorem 1, we have to consider the set $\mathcal{S}$ of times (absolute deadlines) for the computation of $h(t)/t$ in the time interval $[5, 1001)$, where $\mathcal{S}$ is given by:

$$\begin{aligned}\mathcal{S} = &\{5 + 7\,k_1,\ k_1 \in \{0, \ldots, 142\}\} \cup \\ &\{7 + 11\,k_2,\ k_2 \in \{0, \ldots, 90\}\} \cup \\ &\{10 + 13\,k_3,\ k_3 \in \{0, \ldots, 76\}\}\,.\end{aligned}$$

In this example, there are $s = 281$ elements in S.

Applying the linear programming approach on LP1:

The simplex algorithm is applied on the Linear Programming problem LP1, for any time $t_i \in \mathcal{S}$, starting from time t_s down to time t_1 (to optimize the computation). We obtain the following set $\mathcal{S}$ after removing all the unnecessary constraints, maximizing $h(t)/t$ for any set of WCETs $X = \{x_1, x_2, x_3\} \in \mathbb{R}^{+3}$.

$$\mathcal{S} = \{5, 7, 10, 12, 40\}.$$

From theorem 1, we therefore have:

$$\begin{aligned}load(\tau_{(X,T,D)}) = Max\left\{U, Sup_{t\in\mathcal{S}}\left\{\frac{h(t)}{t}\right\}\right\} = Max\Bigg\{&\frac{x_1}{7} + \frac{x_2}{11} + \frac{x_3}{13}, \\ &\frac{x_1}{5}, \frac{x_1 + x_2}{7}, \frac{x_1 + x_2 + x_3}{10}, \frac{2\,x_1 + x_2 + x_3}{12}, \frac{6\,x_1 + 4\,x_2 + 3\,x_3}{40}\Bigg\}.\end{aligned}$$

4.2. Computing of the maximum allowance with EDF
We now show how to compute in theorem 2 the maximum extra duration that can be granted to a single sporadic task $\tau_i(C_i, T_i, D_i)$, with a sensitivity analysis valid for arbitrary deadlines. This duration is denoted as the allowance of $\tau_i(C_i, T_i, D_i)$ in [42] and is the maximum value of A_i such that the task set $\tau_i(C_i + A_i, T_i, D_i) \cup \{\cup_{j=1, j\neq i}^{n} \tau_j(C_j, T_j, D_j)\}$ is schedulable, assuming that $\cup_{j=1, j\neq i}^{n} \tau_j(C_j, T_j, D_j)$ is schedulable. When $A_i < 0$, we know precisely the maximum acceptable duration of τ_i: $C_i + A_i$. This property is used to compute the maximum acceptable portion for a task in section 5.1, assuming that the tasks already assigned to the processor are schedulable.

In the following theorem 2, τ/τ_i denotes the set of tasks in τ with τ_i excluded.

Theorem 2. *Let τ/τ_i be a schedulable task set with EDF. The maximum allowance A_i of a sporadic task $\tau_i(C_i, T_i, D_i)$ is the solution of* $A_i = min(min_{t \geq D_i}\{\frac{t}{1+\lfloor\frac{t-D_i}{T_i}\rfloor}(1-\frac{h(t)}{t})\}, (1-U)T_i)$.

Proof: The allowance of $\tau_i(C_i, T_i, D_i)$ must satisfy two conditions: (i) $\forall t \geq D_i$, $h(t) + (1 + \lfloor\frac{t-D_i}{T_i}\rfloor)A_i \leq t$ and (ii) $U + \frac{A_i}{T_i} \leq 1$.

The first condition (i) leads to: $\forall t \in \mathcal{S}, A_i \leq \frac{t}{1+\lfloor\frac{t-D_i}{T_i}\rfloor} \times (1 - \frac{h(t)}{t})$. It follows that $A_i \leq min_{t \geq D_i}\{\frac{t}{1+\lfloor\frac{t-D_i}{T_i}\rfloor}(1 - \frac{h(t)}{t})\}$.

The second condition (ii) leads to $A_i \leq (1 - U)T_i$. A_i is thus the minimum value satisfying both conditions. ■

Theorem 2 is an adaptation of a result presented in [45] where it is shown that $A_i = min_{t \geq D_i}\{\frac{t}{1+\lfloor\frac{t-D_i}{T_i}\rfloor}(1 - \frac{h(t)}{t})\}$. But if we consider for example a task set composed of only one task $\tau_1 = \{C_1 = 20, T_1 = 100, D_1 = 120\}$. We have for τ_1, $min_{t \geq D_1}\{\frac{t}{1+\lfloor\frac{t-D_1}{T_1}\rfloor}(1 - \frac{h(t)}{t})\} = 100$ whose maximum value is obtained for time $t = 120$. Nevertheless, in that case, $(1 - U)T_1 = 80$. Hence, $A_1 = 80$. Hence, the computation of A_i given in [45] is not valid for arbitrary deadlines.

4.3. Minimum acceptable deadline with EDF

We now revisit the result of [46] to compute the minimum acceptable deadline of a task scheduled with EDF and present a modified version in function computeDmin that addresses some problems we detail in the following. This result is used in section 5.1 to compute the minimum acceptable deadline for the EDF-MLD-Dmin heuristic.

We present some explanations and counter examples showing our corrections on the algorithm given in [46] by Balbastre et al. In function computeDmin, lines 8 to 12 have been added and line 7 has been modified with respect to the original algorithm.

Error with "$t = sT_i + D_i$". The algorithm proposed by Balbastre et al. does not appear to include the case where we have $h(t) = t$ at a time $t = sT_i + D_i$, the deadline of the considered task τ_i, with s being a positive integer. Here, the deadline of τ_i should not be reduced and should be kept equal to D_i.

For this specific case, the algorithm proposed in [46] does not seem to be perfectly clear since we do not successfully know if these times have to be considered or not. However, in both cases, we show with some examples that the condition is not correct.
If we consider those times, the algorithm will give task τ_i a deadline equal to $h(t) + C_i - sT_i = D_i + C_i > D_i$ leading to a higher deadline than D_i, not the minimum (i.e. D_i). If we do not consider those times, the algorithm can provide deadlines that are too small. We have corrected the algorithm by adding lines 8 to 12.

A counter example based on the task set is given in Table 2.

If we compute the minimum acceptable deadline for task τ_3 with the original algorithm we have:

1. For the first iteration, we initialize $D_3^{min} = C_3 = 44$

Function computeDmin : compute the minimum deadline for a task τ_i

Input: Task $\tau_i(C_i, T_i, D_i)$, Task set $\tau_{(C,T,D)}$
Output: Minimum deadline for τ_i

1 $P =$ LCM of periods of tasks in $\tau_{(C,T,D)}$;
2 $deadline = 0$;
3 $k_i = \lceil \frac{P}{T_i} \rceil$;
4 $D_i^{min} = 0$;
5 **for** *s=0* **to** k_i*-1* **do**
6 $\quad t = sT_i + D_i$;
7 $\quad deadline = \max(C_i, h(sT_i + C_i) + C_i - sT_i)$;
8 $\quad$ **if** *t = h(t)* **then**
9 $\quad\quad D_i^{min} = D_i$;
10 $\quad\quad$ exit-for;
11 $\quad$ **else**
12 $\quad\quad t = t - 1$;
13 $\quad\quad$ **while** $t > sT_i + C_i$ **do**
14 $\quad\quad\quad$ **if** t *is an absolute deadline of a task* τ_j *in* $[0, P]$ **then**
15 $\quad\quad\quad\quad$ **if** $t - h(t) < C_i$ **then**
16 $\quad\quad\quad\quad\quad deadline = h(t) + C_i - sT_i$;
17 $\quad\quad\quad\quad\quad exitWhileLoop$;
18 $\quad\quad\quad\quad$ **end if**
19 $\quad\quad\quad$ **end if**
20 $\quad\quad\quad t = t - 1$;
21 $\quad\quad$ **end while**
22 $\quad$ **end if**
23 $\quad D_i^{min} = max(D_i^{min}, deadline)$;
24 **end for**
25 **return** D_i^{min};

2. We search for absolute deadlines in the interval $[C_3; D_3] = [44; 54]$. The only time to consider is $t = 54$ which is the deadline of task τ_3.
3. At time $t = 54$:
   - if we consider this time, the new minimum deadline will be equal to $D_3^{min} = h(44) + C_3 - 0 \times T_3 = 54 + 44 + 0 = 98$ which is higher than the actual deadline ($D_3 = 54$).
   - if we do not consider this time, the final minimum deadline will remain equal to the initial value $D_3^{min} = C_3 = 44$, which leads to an unfeasible task set with a load equal to 1.2272.

With our modification of lines 8 to 12, we find that $h(54) = 54$, thus $h(D_3) = D_3$ and we fix the minimum possible deadline to $D_3^{min} = D_3 = 54$. Any reduction of this deadline will lead to a load higher than 1.

Error with the initialization "$deadline = C_i$*".* In the original algorithm, the variable *deadline* is initialized as $deadline = C_i$. At line 7 of our algorithm, we have replaced this

Task	**C**	**T**	**D**
τ_1	10	54	16
τ_2	12	97	91
τ_3	44	88	54

Table 2: Task set τ considered as a counter example for the case $t = sT_i + D_i$

initialization with $deadline = \max(C_i, h(sT_i + C_i) + C_i - sT_i)$.

A counter example based on a task set is given in Table 3.

Task	**C**	**T**	**D**
τ_1	10	55	16
τ_2	12	88	80
τ_3	44	88	80

Table 3: Task set τ considered as a counter example for the initialization $deadline = C_i$

Computing the minimum acceptable deadline for task τ_3 with the original algorithm leads to the following:

1. For the first iteration, we initialize $D_3^{min} = C_3 = 44$
2. We search for absolute deadlines in the interval $[C_3; D_3] = [44; 80]$. We have to consider times $t = 80$, deadline of tasks τ_2 and τ_3 and $t = 71$, the second deadline of task τ_1.
3. At time $t = 80$:
   - if we consider this time, the new minimum deadline of τ_3 will be equal to $D_3^{min} = h(80) + C_3 - 0 \times T_3 = 76 + 44 + 0 = 120$ which is higher than the actual deadline ($D_3 = 80$).
   - if we do not consider this time, the minimum deadline remains equal to the initial value $D_3^{min} = C_3 = 44$ which leads to an unfeasible task set with a load equal to 1.2272.
4. At time $t = 71$:
   - We have $h(71) = 20$ and $t - h(t) = 71 - h(71) > C_3 = 44$ thus this time is ignored and the minimum acceptable deadline for τ_3 remains equal to 44 or 120 according to the previous point.

With our modification of line 7, we will initialize $D_3^{min} = \max(C_3, h(0 \times T_3 + C_3) + C_3 - 0 \times T_3) = \max(44, h(44) + 44) = \max(44, 10 + 44) = 54$. Any reduction of this deadline will lead to a load greater than 1.

5. Two approaches for the EDF semi-partitioned scheduling

In this section we present two solutions for the EDF semi-partitioned scheduling problem:

- The first solution, presented in subsection 5.1, is an extension of the solution proposed by Kato & al. ([3]), for the portioned scheduling of tasks. We show that the solution in [3] corresponds to the case where all the processors apply jitter cancellation before migrating the job of a task at a local deadline function of the deadline of the task. With jitter cancellation, the job inter-arrival times are identical on all the processors executing a portion of a job. Hence, the feasibility conditions done on each processor are independent and no jitters should be taken into account in the feasibility conditions. We establish a generic feasibility condition valid for different local deadline assignment heuristics. We then propose a new local deadline assignment based on the minimum acceptable local deadline for a task on all the processors executing it.

- The second solution, presented in section 5.2, is based on job partitioning. With job partitioning, migrations only occur at job boundaries. We propose executing a task on $s \leq m$ processors according to a Round Robin job migration of jobs between the s processors. A processor therefore executes the job of a task 1 time out of s.

In both cases, we first try to assign the tasks according to a partitioning heuristic. For the simulations in section 6, we consider two heuristics: First-Fit Decreasing (FFD) and Worst-Fit Decreasing (WFD). We use the density parameter $\lambda_i = \frac{C_i}{min(T_i,D_i)}$ for the decreasing policy. Any task set $\tau_{(C,T,D)}$ is indexed by decreasing value of $\lambda_i = \frac{C_i}{min(T_i,D_i)}$.

We use a necessary and sufficient condition to assign the tasks to a processor: $\forall \pi_k, k \in [1, m], load(\tau^{(k)}) \leq 1$. $load(\tau^{(k)})$ is equal to $load(\tau)$ with WCETs of tasks not assigned to processor π_k equal to 0.

We use the same approach as that given in [44] to reduce the complexity of the computation of $load(\tau)$, by first applying a linear programming approach (see section 4.1) to reduce the number of elements characterizing the load function, before using the $load(\tau_{(X,T,D)})$ function in the partitioning heuristics.

5.1. Portioned semi-partitioned scheduling

With portioned semi-partitioned scheduling, the job of a task $\tau_i(C_i, T_i, D_i)$ that cannot be executed on a single processor is portioned and executed by subtasks on a set of processors. We investigate the solution where a local deadline is assigned to each subtask. When the WCET of a task is portioned to be executed on $2 \leq s_i \leq m$ processors, this results in $s_i - 1$ migrations. The migration occurs w.r.t to the release time of a job, at the local deadline of the job.

The problems considered for portioned semi-partitioning are therefore:

- To find a local deadline assignment heuristic for the task on each processor executing the task.

- To find a portioning heuristic for the WCET of a task.

These two heuristics are mutually dependent (the duration of a portion depends on the local deadline chosen). Kato & al. ([3]) present a portioning heuristic that minimizes the number of processors required to execute a task by assigning the maximum possible duration to the WCET of a portion while preserving the schedulability of the task. We adopt the same principle for the portioned heuristic we study as it minimizes the number of processors required to execute a task, and hence reduces the number of migrations. We revisit the solution proposed by Kato & al. in [3]. We use theorem 2 to compute the WCET of a portion from the allowance of a task.

As regards the local deadline assignment heuristic, we describe different local deadline assignment heuristics in section 5.1.1. Using simulations, we study the performance of different local deadline assignment strategies in section 6.

If we do not control the migration times, a task can experience release jitter that increases with the number of processors executing the task. This problem is well known in distributed systems. The holistic approach ([47]) has been considered to compute the worst case end-to-end response time of a sporadic task, taking into account the jitter resulting from all the visited nodes. With this approach, the worst case response times on each node are not independent and the jitter increases with the number of processors used.

We propose a solution based on jitter cancellation. With jitter cancellation, we cancel the release jitter of jobs before migrating them. The job arrival pattern is therefore the task arrival pattern (sporadic) on all processors. We are then able to apply a uniprocessor feasibility condition on any processor that only depends on the subtasks assigned to the processors. [48] propose, for example, this technique in the context of distributed systems.

Definition 1. *With* jitter cancellation*, a job of task $\tau_i(C_i^{(k)}, T_i, D_i)$ released at time t_i on a processor π_k will do a migration at time $t_i + D_i^{(k)}$, where $D_i^{(k)}$ is the local deadline of τ_i on π_k. The duration of $D_i^{(k)}$ is chosen to be at least equal to the worst case response time of τ_i on π_k.*

The generic scheduling we propose is denoted as EDF-MLD-* (EDF scheduling with Migration at the Local Deadline of the job of a subtask). We use the notation EDF-MLD-WM for the solution proposed by Kato & al. [3].

Property 4. *Jitter cancellation enables us to analyze the schedulability of portioned semi-partitioned systems on each processor independently.*

Proof: With a migration as proposed in definition 1, we cancel the possible release jitter on the following processors. The worst case response time of a job on a processor (or a portion of a job) thus only depends on the jobs executed on the processors with no release jitter. With jitter cancellation, the recurrence of subtasks is in a identical pattern on the different processors.

The job release times of a subtask on a processor therefore follow the sporadic arrival times of the tasks it comes from as the migration does not constrain the worst case scenario on each processor (any release time scenario is possible). ■

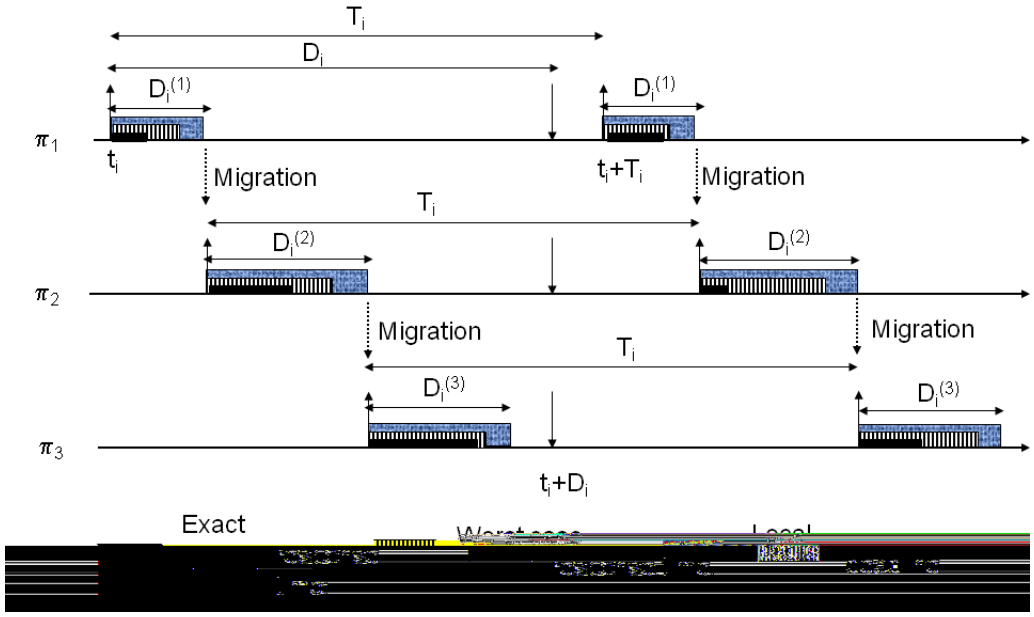

Figure 2: Migration at local deadlines

Figure 2 illustrates the principle of migration at the local absolute deadline of jobs. Theorem 3 is a generic feasibility condition for portioned EDF scheduling when migrations occur on processor π_k for a job of subtask of τ_i released at time t_i at time $t_i + D_i^{(k)}$.

Theorem 3. *Let τ_i be a task executed on $2 \leq s \leq m$ processors by portioned EDF scheduling. Suppose that the migration of a subtask of τ_i released at time t_i on $\pi_k \in [1, s-1]$ occurs at time $t_i + D_i^{(k)}$. A necessary and sufficient feasibility condition for a task τ_i is:*

- $\sum_{i=1}^{s} C_i^{(k)} = C_i$
- $\forall k \in [1, s]$, *the local deadlines $D_i^{(k)}$ are met.*
- $\sum_{i=1}^{s} D_i^{(k)} \leq D_i$

Proof: The first condition ensures that the task can be entirely executed on s processors. The second condition checks on each processor involved π_k that the local deadlines assigned to a subtask of task τ_i are met. This can be done by a classical uniprocessor feasibility condition, based on the worst case response time of a job for fixed priority scheduling [49] or based on the load($\tau_{(C,T,D)}$) for EDF (see section 4). The third condition checks that the sum of local deadlines of τ_i on the s processors is at most equal to the deadline D_i. ■

Using algorithm 1, we now describe the solution used to decide on the schedulability of a sporadic task τ_i for a portioned semi-partitioned scheduling. As in [3], we first try to assign as many tasks as possible with a classical First-Fit or Worst-Fit heuristic. The goal is to keep the number of migrating tasks to a minimum. Algorithm 1 is called for each task one by one, when a classical FF or WF heuristic fails to schedule a task on only

one processor. The first step is to compute local deadline $D_i^{(k)}$ and allowance $A_i^{(k)}$ on each processor π_k, $k \in [1, m]$, for the subtask of task τ_i we want to assign. We then use the Sort-processors() function to sort the processors list either by decreasing order of allowance or by increasing processor utilizations, depending on the local deadline assignment heuristic used to assign local deadlines (see section 5.1.1). The maximum duration of the portion that can *be assigned to the first* s_i processors for τ_i is equal to $\sum_{k=1}^{s_i} C_{i,max}^{(k)}$, where $C_{i,max}^{(k)} = C_i + A_i^{(k)}$. The duration $A_i^{(k)} < 0$ as by construction, τ_i cannot be executed on $s_i - 1$ processors. $A_i^{(k)}$ is computed by the $computeAllowance(C, T, D, Processor)$ function that can be implemented from theorem 2. We then check if the sum of the WCETs that could be given to τ_i on s_i processors is at least equal to C_i. If this is the case then τ_i can be assigned to the s_i processors, otherwise, s_i is increased by 1 and so forth until reaching $s_i = m$. If we proceed the execution of the algorithm until the last line (line 17), this means that task τ_i cannot be assigned to m processors.

Algorithm 1: Generic algorithm for the assignment of τ_i on processors

Input: Processor set π, Task set $\tau_{(C,T,D)}$, Task $\tau_i(C_i, T_i, D_i)$ in $\tau_{(C,T,D)}$
Data: s_i, k are Integers

```
1  for s_i=2 to m do
2      for k=1 to s_i do
3          D_i^(k) = computeLocalDeadline();
4          A_i^(k) = computeAllowance(C_i, T_i, D_i^(k), π_k);
5      end for
6      Sort-processors(s_i);
7      [Sort processors by decreasing allowances or increasing processor
       utilizations];
8      for k = 1 to s_i do
9          C_{i,max}^(k) = C_i + A_i^(k);
10     end for
11     if Σ_{j=1}^{s_i} C_{i,max}^(j) ≥ C_i then
12         [Task τ_i can been assigned to s_i processors];
13         assignTask(τ_i, s_i);
14         exitProgram();
15     end if
16 end for
17 notFeasible(τ_i);
```

The result of this algorithm, when every task can be scheduled is:

- For each task, the set of processors executing it;
- The WCETs of the portions of each subtask on each processor
- The local deadlines used for each task on each processor

We now establish the feasibility conditions for the portioned semi-partitioning problem in a particular case of migration at fair local deadlines. We consider the case where a task

that cannot be executed on a single processor is portioned on a set of $s_i > 1$ processors. Migration occurs at the local deadlines equal to $\frac{D_i}{s_i}$ w.r.t. the job release time for all the s_i processors. This is the case considered by Kato & al. in [3]. We show that the load($\tau_{(C,T,D)}$) function can help provide a generic feasibility condition in that case.
The notations used w.r.t. the portioned partitioning problem are the following:

- $\tau_{(X,T,D)}$ is a sporadic task set, where $X = (x_1, \ldots, x_n)$ is a set of WCETs variables, T and D a set of fixed periods and deadlines.
- $\tau^{(k)}_{(X^{(k)},T,D^{(k)})}$ denotes the set of tasks in $\tau_{(X,T,D)}$ assigned to processor π_k with WCET portions variables defined in the set $X^{(k)} = (x_1^{(k)}, \ldots, x_n^{(k)})$, a set of fixed periods T, a set of local deadlines deadlines $D^{(k)}$. By convention, $x_i^{(k)} = 0$ means that no portion of task $\tau_i(x_i, T_i, D_i^{(k)})$ is assigned to processor π_k.
- load($\tau^{(k)}_{(X^{(k)},T,D^{(k)})}$) denotes the load function of task set $\tau^{(k)}_{(X^{(k)},T,D^{(k)})}$ obtained from task set $\tau_{(X,T,D)}$ assigned to processor π_k.
- $\frac{D}{p}$ denotes the set of deadlines where all the deadlines in D are divided by $p \in \mathcal{N}, p > 1$.

Consider a task set composed of $m \times n$ tasks: $\tau_{(X_1,T,D)} \cup \tau_{(X_2,T,\frac{D}{2})} \cup \ldots \cup \tau_{(X_m,T,\frac{D}{m})}$. This task set is composed of all the tasks in τ with all possible fair local deadlines. X_i denotes a set of n WCETs variables and $X_i^{(k)}$ denotes a set of portions of WCETs for the tasks assigned to processor π_k.

We now propose a feasibility condition for the portioned EDF semi-partitioned with fair local deadline assignment in theorem 4. When considering a particular portioning of the tasks, $C^{(k,s)}$ denotes in theorem 4 the set of WCETs portions of all the tasks on processor π_k when portions have a corresponding deadline in D/s (the WCET of a portion is equal to 0 if the task is not assigned).

Theorem 4. *Let $\tau_{(C,T,D)}$ be a task set scheduled using EDF on m processors according to an arbitrary portioned semi-partitioning with a fair local deadline assignment. A necessary and sufficient feasibility condition for the EDF partitioned scheduling is:* $\forall k \in [1, m]$, $load(\tau_{(X_1^{(k)}=C^{(k,1)},T,D)} \cup \tau_{(X_2^{(k)}=C^{(k,2)},T,\frac{D}{2})} \cup \ldots \cup \tau_{(X_m^{(k)}=C^{(k,m)},T,\frac{D}{m})}) \leq 1$.

Proof: The load function is computed for $m \times n$ tasks in all configurations of fair deadlines. When considering a particular portioning of the tasks in the set $C^{(k,s)}$ on processor π_k having corresponding deadlines in D/s, the WCET of a portion is equal to 0 if a portion of the task is not assigned to processor π_k. This condition checks that the load corresponding to the tasks really assigned to processors is at most equal to 1. ■

By applying the linear programming approach LP1 to reduce the number of elements in $\mathcal{S}$ that are needed to compute the load function, we obtain a generic feasibility condition to decide on the schedulability of a fair portioned semi-partitioning.

5.1.1. Local deadline assignment heuristics

Several local deadline assignment heuristics have been proposed for the local deadlines. We adapt the proposed heuristics to the context of a task τ_i being executed on a set of s processors according to a semi-partitioned scheduling.

We cite three local deadline assignments proposed in [50] (in the context of a network composed of heterogeneous segments, which corresponds to the case of uniform multiprocessors). The performance of these local deadline assignments is studied in section 6. For any task τ_i, the local deadline $D_i^{(k)}$ on processor $\pi_k, k = 1, \ldots, s$ can be defined as follows for EDF scheduling:

- The first method is the *fair local deadline assignment* and corresponds to EDF-MLD-WM scheduling in the simulations: assign for a subtask of τ_i a local deadline $\frac{D_i}{s}$. This corresponds to the solution proposed by Kato $\&$ al in [3].
- The second solution is the *fair deadline and fair WCET assignments* and corresponds to EDF-MLD-Fair scheduling in the simulations. This algorithm is the simplest since it distributes WCET and deadlines fairly among the processors.
- The third solution assigns a local deadline proportional to the sum of the processor utilizations, referred to in the simulation section EDF-MLD-U, that is: $D_i^{(k)} = \frac{U_k}{\sum_{j=1}^{s} U_j} D_i$, where U_k is the processor utilization of all the tasks and subtasks already assigned to π_k including the processor utilization of the subtask of τ_i on π_k.

5.1.2. The EDF-MLD-Dmin scheduling

We now consider a fourth local deadline assignment denoted as EDF-MLD-Dmin based on an optimization of EDF-MLD-WM. The principle of this local deadline assignment is to consider the deadline margin on each local deadline $\frac{D_i}{s}$ after the maximum WCET computation. This margin is computed according to the minimum local deadline computation presented in section 4.3. If a subtask can be assigned to processor π_k, the minimum local deadline ($D_{i_{min}}^{(k)}$) is computed and the difference $D_{reserve} = \frac{D_i}{s} - D_{i_{min}}^{(k)}$ is given to the following processor in order to increase the allowance of the portion of WCET that can be assigned to it. The next processor can use a deadline equal to $\frac{D_i}{s} + D_{reserve}$ and is likely to increase its allowance. A detailed procedure is given in function tryToSplitAndAssignTask. We study the performance of this local deadline assignment heuristic in section 6.

Function tryToSplitAndAssignTask : Assigns local deadlines w.r.t. to EDF-MLD-Dmin heuristic

```
1  [m is the number of processors in π];
   Input: Processor set π, Task τ_i(C_i, T_i, D_i)
   Output: True if the task has been assigned, False otherwise
2  D_reserve ← 0;
3  for s_i=2 to m do
4      for k=1 to s_i do
5          for l=1 to m do
6              D_i^(l) = D_i/s_i + D_reserve;
7              A_i^(l) = computeAllowance(C_i, T_i, D_i^(l), π_l);
8          end for
9          [π_g is the processor with the greatest allowance A_i^(g) = max_{l=1...m}(A_i^(l)) in
           order to reduce the number of subtasks]
10         C_{i,max}^(k) = C_i + A_i^(g);
11         D_i^(k) = computeDmin(C_{i,max}^(k), T_i, D_i^(k), π_g);
12         D_reserve = D_i/p − D_i^(k);
13     end for
14     if Σ_{j=1}^{s_i} C_{i,max}^(j) ≥ C_i then
15         [Task τ_i can be assigned to s_i processors];
16         [If D_reserve > 0 it will be re-assigned uniformly];
17         assignTask(τ_i, s_i);
18         exitSuccess();
19     end if
20 end for
21 exitFail();
```

5.2. Job partitioning with Round Robin Job Migration

We now present a semi-partitioning heuristic based on job migrations at job boundaries (see section 3.3), that consists of assigning the subtasks of a task to a set of processors in the following way.

Definition 2. *Let τ_i be a task assigned to a set of $s \leq m$ processors according to a job partitioning heuristic. The job partitioning is a Round Robin Job Migration heuristic if the job migrations of τ_i follow a Round-Robin pattern: first on π_1, then on π_2, ..., then on π_s, then on π_1 and so forth.*

When tasks are strictly periodic and the first release of τ_i occurs at time 0 on processor π_1, we obtain the following pattern of arrivals on the s processors: the k^{th} job of τ_i is released at time $(k-1)T_i$ on processor $((k-1)Mod\ s)+1$, where Mod stands for the modulo function.
This leads to releasing the jobs of τ_i on each processor executing it, with a period equal to $s \times T_i$.

Consider the task set $\tau_{(X_1,T,D)} \cup \tau_{(X_2,2T,D)} \ldots \cup \tau_{(X_m,mT,D)}$. This task set is composed of $m \times n$ tasks having periods equal to $T, 2T, 3T, \ldots, mT$, deadlines equal to the deadlines in D and WCETs defined by the sets of WCET variables $X_1, X_2, \ldots, X_m$.
Theorem 5 provides a necessary and sufficient feasibility condition for the schedulability of a sporadic task set scheduled with the semi-partitioned EDF scheduling on m processors according to a EDF-RRJM scheduling. $C^{(k,s)}$ denotes in theorem 5 the set of WCETs of all the tasks on processor π_k when tasks have a corresponding period in sT (the WCET of a subtask is equal to 0 if not assigned).

Theorem 5. *Let $\tau_{(C,T,D)}$ be a task set scheduled with the semi-partitioned EDF-RRJM scheduling on m processors. A necessary and sufficient feasibility condition for EDF-RRJM semi-partitioned scheduling is: $\forall k \in [1,m], load(\tau_{(X_1^{(k)}=C^{(k,1)},T,D)} \cup \tau_{(X_2^{(k)}=C^{(k,2)},2T,D)} \cup \ldots \cup \tau_{(X_m^{(k)}=C^{(k,m)},mT,D)}) \leq 1$.*

Proof: The proof is similar to that provided in theorem 4. ■

By applying the linear programming approach LP1 to reduce the number of elements in $\mathcal{S}$ that are needed to compute the load function, we obtain a generic feasibility condition to decide on the schedulability for the semi-partitioned scheduling with Round Robin job migrations.

5.3. Example of the application of the load function

For this example, we generate the following task set τ described in table 4:

C	T	D
x_1	70	60
x_2	110	72
x_3	130	84

Table 4: The task set τ considered

In the case of $m = 4$, we generate task sets $\tau_{(C,2T,D)}$, $\tau_{(C,3T,D)}$ and $\tau_{(C,4T,D)}$ equivalent to τ with the period of each task multiplied by respectively 2, 3 and 4. Finally, we define $\tau_{ST} = \{\tau_{(C,T,D)} \cup \tau_{(C,2T,D)} \cup \tau_{(C,3T,D)} \cup \tau_{(C,4T,D)}\}$. In the same way, we generate the task sets where the deadlines and WCETs are respectively divided by 2, 3 and 4 leading to the task sets: $\tau_{(\frac{C}{2},T,\frac{D}{2})}$, $\tau_{(\frac{C}{3},T,\frac{D}{3})}$ and $\tau_{(\frac{C}{4},T,\frac{D}{4})}$. Finally we generate the task set $\tau_{SD} = \tau_{(C,T,D)} \cup \tau_{(\frac{C}{2},T,\frac{D}{2})} \cup \tau_{(\frac{C}{3},T,\frac{D}{3})} \cup \tau_{(\frac{C}{4},T,\frac{D}{4})}$
Table 5 shows the results of the simplex algorithm on these task sets. The list of the elements to consider in order to compute the *load(τ)* function is given in the third column. We can see the performance of the simplex algorithm applied to LP1 optimization problem (see section 4.1).

Let us do a numeric application to verify property 3 on $s = 2$ processors i.e. that $load(\tau_{(C,2T,D)}) = load(\tau_{(\frac{C}{2},T,\frac{D}{2})})$. We choose $x_1 = 20$, $x_2 = 48$ and $x_3 = 36$.

Table 5: Elements in $\mathcal{S}$ to consider in the *load* computation according to the result of the simplex algorithm

Task sets	**Number of elements in [D_{min}, P]**	**With simplex algorithm**	
		Number of elements	**List of elements**
$\tau_{(C,T,D)}$	311	7	60, 72, 84, 130, 200, 410, 622
$\tau_{(C,2T,D)}$	311	3	60, 72, 84
$\tau_{(C,3T,D)}$	311	3	60, 72, 84
$\tau_{(C,4T,D)}$	311	3	60, 72, 84
τ_{ST}	3732	7	60, 72, 84, 130, 200, 410, 622
$\tau_{(C/2,T,D/2)}$	311	3	30, 36, 42
$\tau_{(C/3,T,D/3)}$	311	3	20, 24, 28
$\tau_{(C/4,T,D/4)}$	311	3	15, 18, 21
τ_{SD}	1223	22	15, 18, 20, 21, 24 28, 30, 36, 42, 60 72, 84, 85, 90, 100 130, 134, 200, 410 411, 480, 622

As expected, according to Table 5 we have :

$$load(\tau_{(C,2T,D)}) = \max(\frac{x_1}{70} + \frac{x_2}{110} + \frac{x_3}{110}, \frac{x_1}{60}, \frac{x_1 + x_2}{72}, \frac{x_1 + x_2 + x_3}{84}) = \frac{104}{84} = \frac{26}{21} \quad (8)$$

$$load(\tau_{(\frac{C}{2},T,\frac{D}{2})}) = \max(\frac{\frac{x_1}{2}}{70} + \frac{\frac{x_2}{2}}{110} + \frac{\frac{x_3}{2}}{110}, \frac{\frac{x_1}{2}}{30}, \frac{\frac{x_1}{2} + \frac{x_2}{2}}{36}, \frac{\frac{x_1}{2} + \frac{x_2}{2} + \frac{x_3}{2}}{42}) = \frac{52}{42} = \frac{26}{21} \quad (9)$$

6. Experimental results for EDF

In this section, we evaluate the effectiveness of the following semi-partitioned scheduling algorithms, using the local deadline assignments described in section 5.1.1.

- EDF-MLD-WM referred to as EDF-WM in [3], that proposes a fair local deadline assignment and full portions.
- EDF-MLD-Fair derived form EDF-WM that proposes a fair local deadline assignment and fair WCETs (deadlines and WCETs divided by the number of processors executing the task).
- EDF-MLD-U derived from EDF-WM that assigns local deadlines to subtasks according to processor utilization.
- EDF-MLD-Dmin derived from EDF-WM that assigns local deadlines to subtasks according to the minimum local deadlines of the tasks (see section 4.3).
- EDF-RRJM (EDF with Round Robin Job Migrations)

- The global scheduling RTA (see section 3.2)

All task sets are sorted in decreasing order of density (parameter λ_i in section 2) before computation as it is an optimization for all partitioning algorithms based on EDF ([17]). We then try to assign tasks according to a heuristic: the classical partitioned EDF-FFD (First-Fit Decreasing) [14, 1] or EDF-WFD (Worst-Fit Decreasing) [16]. Finally, the semi-partitioned algorithms are used for tasks that cannot be assigned to a single processor.

The effectiveness of an algorithm is estimated by its *success ratio* and its *density of migrations*.

The *Success ratio* is defined as follows:

$$\frac{number\ of\ successfully\ scheduled\ task\ sets}{number\ of\ submitted\ task\ sets} \tag{10}$$

The *density of migrations* parameter is equal to the sum for all migratory tasks of the ratio between the maximum number of migrations of a task during a time interval equal to its period divided by its period. Thus, for all semi-partitioned heuristics, with migration at a local deadline (denoted as EDF-MLD-*) the maximum number of migrations for a migratory task is equal to the number of portions required to execute a job (supposing that a migration happens at the local deadline of a job, to release the following job). It follows:

$$\sum_{\tau_i \in migratory\ tasks} \frac{number\ of\ portions\ of\ \tau_i}{T_i} \tag{11}$$

For EDF-RRJM, which gives only a migration between each job for any number of subtasks, we have for a migratory task:

$$\sum_{\tau_i \in migratory\ tasks} \frac{1}{T_i} \tag{12}$$

6.1. Simulation Setup

The task generation model used in this paper is based on the model presented in [51]. However, in our case, the task generation is adapted to each type of deadline considered. In the following, $k_i \in \{D_i, T_i\}$ and $\rho_i \in \{\lambda_i, u_i\}$:

- k_i is uniformly chosen from [1,100];
- ρ_i (truncated between 0.001 and 0.999) has the following distributions:
  1. Uniform distribution between $\frac{1}{k_i}$ and 1;
  2. Bimodal distribution: heavy tasks have a uniform distribution between 0.5 and 1, light tasks have a uniform distribution between $\frac{1}{k_i}$ and 0.5; the probability of a task being heavy is $\frac{1}{3}$;
  3. Exponential distribution of mean 0.25;
  4. Exponential distribution of mean 0.50;
  5. Exponential distribution of mean 0.75;

For implicit deadline task sets, $(k_i, \rho_i) = (T_i, u_i)$ and for constrained deadlines $(k_i, \rho_i) = (D_i, \lambda_i)$. In this last case, the period T_i is uniformly chosen from $[D_i, 100]$.

We consider 4-processor and 8-processor identical platforms.

Task systems are generated so that those obviously unfeasible ($U > m$) or trivially schedulable ($m = n$ and $\forall i \in [1, n]$, $u_i \leq 1$, meaning the capacity of one processor of the identical platform) are not considered during the simulations:

- Step1: Initially, we generate a system which contains $m + 1$ tasks and test it.
- Step2: Gradually, we add new tasks to the system and repeat the tests until the whole utilization of the system exceeds m (the capacity of the identical platform).

We generate 1,000,000 task sets uniformly chosen from the five distributions presented and the two deadlines types, implicit or constrained. Thus, the results are representative of a large number of task sets.

We decide to reduce the time granularity (the minimum possible value of each parameter) to 1. Thus, for the simulation, the parameters C_i, T_i, D_i are considered as integers. Considering that the values are discretized according to the clock tick, it is always possible to modify all the parameters to integer values by multiplying them by an appropriate factor. To simplify testing, we used this approach and all the parameters are limited to integer values. This does not imply, however, that the algorithms used and presented in this paper cannot be applied to non-integer values.

6.2. Simulation Results

Now, we show the simulation results, obtained for 4 and 8 processors, in terms of the success ratio in subsection 6.2.1 and then in terms of the density of migration in subsection 6.2.2, under discrete time granularity.

6.2.1. Success Ratio

Figure 3 shows the results of simulations based on the success ratio with 4 processors. Our graphs focus on the range of utilization [3.0;4.0] since all algorithms of partitioned and semi-partitioned approaches implemented in this study have the same performance with a lower utilization. In the same way, figure 4 shows the results obtained with 8 processors and focus on the range [7.0;8.0]. As expected, semi-partitioned approaches become useful for high utilization task sets.
We have carried out a study to compare the behavior of these algorithms with task sets exclusively composed of light tasks (based on task sets generated with an exponential distribution of mean 0.25) or heavy tasks (with an exponential distribution of mean 0.75) but the difference between these results and the general case did not appear significant. We therefore focus on the arbitrary load case in the experiments.

With 4 or 8 processors, the global scheduling feasibility condition RTA is clearly outperformed by all other algorithms based on partitioned scheduling algorithms. Thus, if we

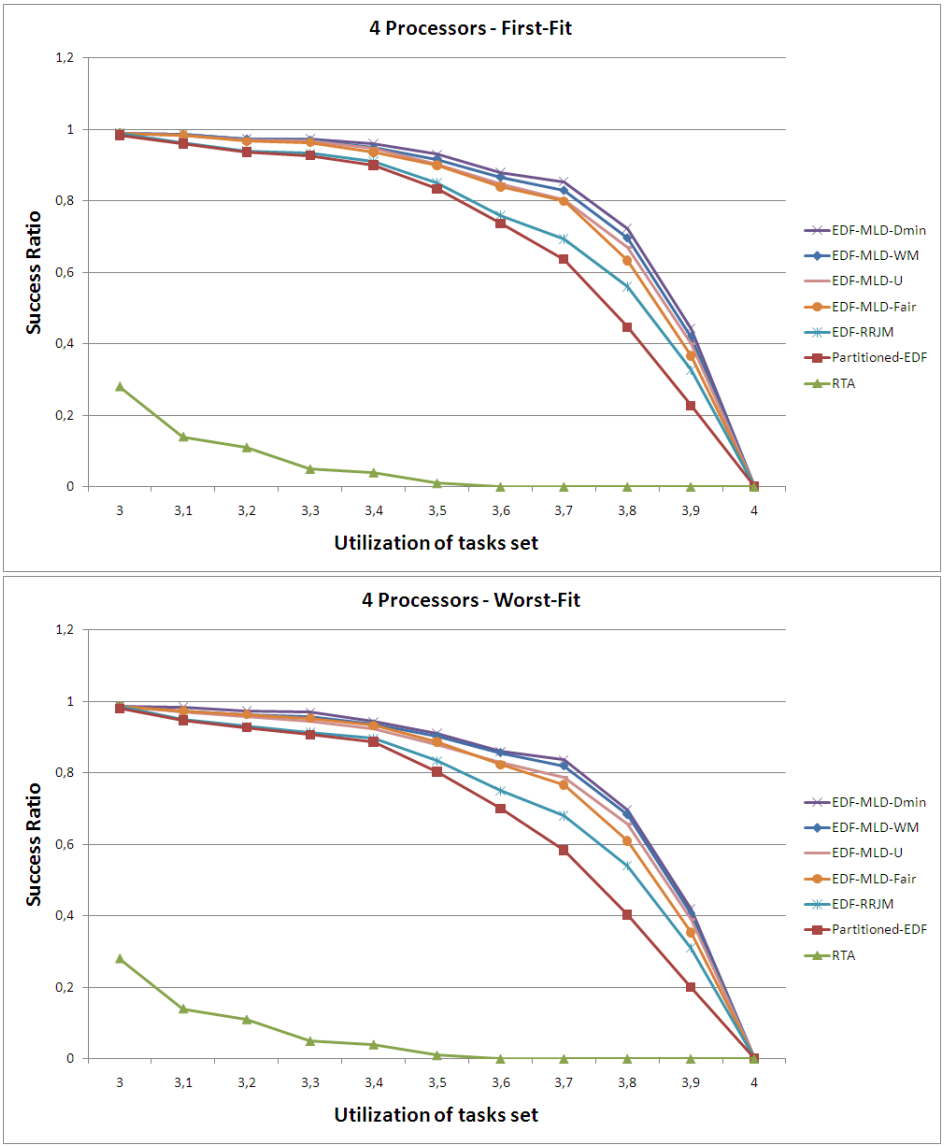

Figure 3: Success Ratio according to utilization of task sets - 4 processors

focus on EDF-based scheduling, the global approach is not as effective as semi-partitioned scheduling, in the current state-of-the-art. Therefore, we will analyze only the difference between semi-partitioned approaches.
Semi-partitioned approaches improve the success ratio of partitioned EDF scheduling. Since WCET portioning is done only when partitioned scheduling algorithms fail to schedule a task on a processor, semi-partitioned algorithms schedule all task sets that are schedulable with partitioned-EDF scheduling algorithms.

We compare for 4 and 8 processors the percentage of the improvement in the success ratio (the difference between the success ratio of semi partitioned EDF and partitioned EDF divided by the success ratio of partitioned EDF times 100) when First-Fit and Worst-Fit heuristics are used.
With 4 processors, the trends obtained with FF and WF are similar. In table 6, we present

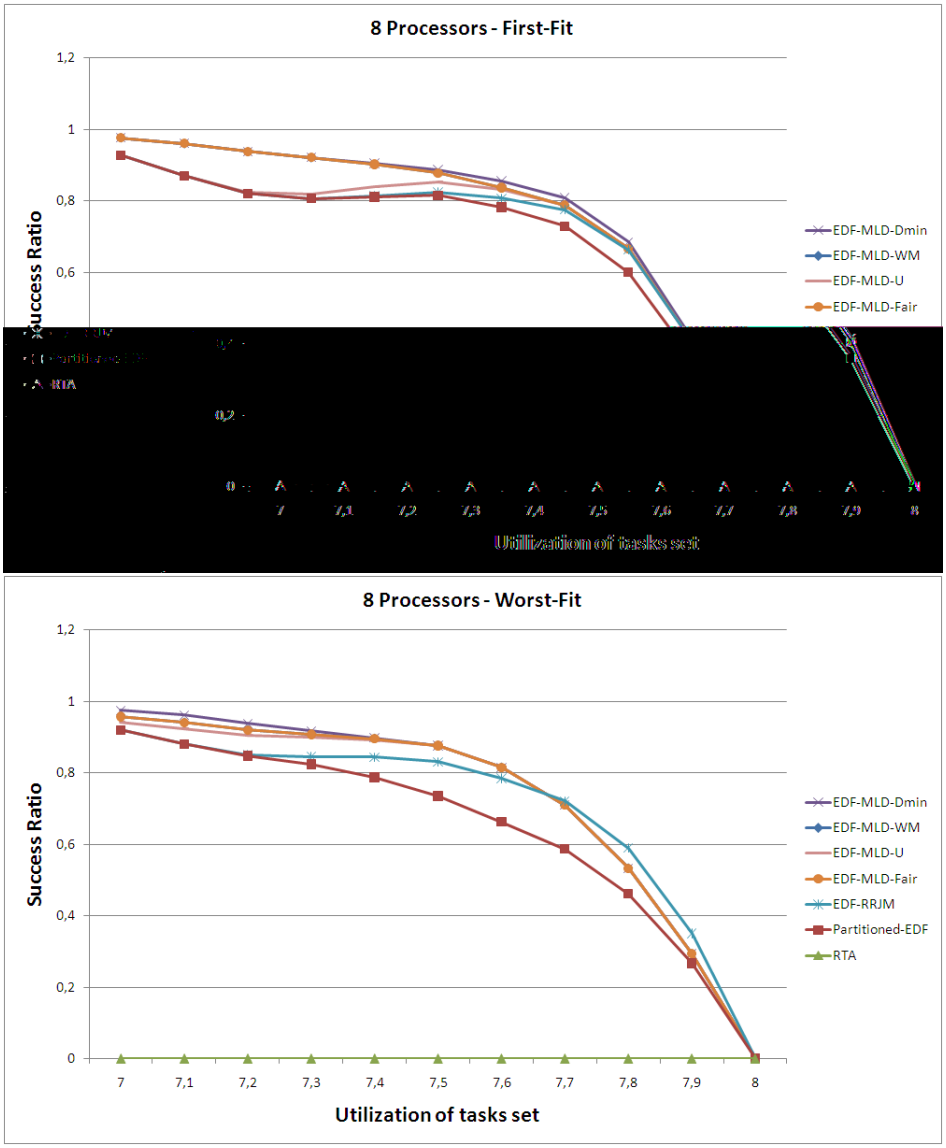

Figure 4: Success Ratio according to processor utilization of task sets - 8 processors

a comparative table of the percentage of the success ratio improvement. We also provide the value of processor utilization (U) for which the percentage of difference is reached. EDF-MLD-Dmin slightly outperforms all the others. The performance of EDF-MLD-WM remains higher than EDF-MLD-U and EDF-MLD-Fair. Thus, the success ratio is clearly proportional to the complexity of computation. In terms of the percentage of improvement, EDF-RRJM outperforms partitioned-EDF scheduling algorithms: between 1% to 10% of task sets were found schedulable for a processor utilization of less than 3.6 and up to 55% for task sets with a higher utilization. In the same range, EDF-MLD-WM improves the schedulability respectively by 1% to 20% and up to 103.3% for high utilization. Finally, EDF-MLD-Dmin reaches an improvement of 110% which represents a gain of about 5.56% compared to EDF-MLD-WM for First-Fit. As we expected, this approach become interesting for task sets with a very high total utilization.

Scheduling	First-Fit				Worst-Fit			
	U	SR1	SR2	Improv.	U	SR1	SR2	Improv.
EDF-RRJM	3.9	0.2267	0.3267	44.11%	3.9	0.2000	0.3100	55.00%
EDF-MLD-Fair	3.9	0.2267	0.3667	61.76%	3.9	0.2000	0.3533	76.65%
EDF-MLD-WM	3.9	0.2267	0.4200	85.27%	3.9	0.2000	0.4067	103.35%
EDF-MLD-DMIN	3.9	0.2267	0.4433	95.54%	3.9	0.2000	0.4200	110.00%
EDF-MLD-U	3.9	0.2267	0.4000	76.44%	3.9	0.2000	0.3900	95.00%

Table 6: Best improvement, in %, of the success ratio of each algorithm (SR2) w.r.t. partitioned EDF (SR1) vs U for 4 processors

Scheduling	First-Fit				Worst-Fit			
	U	SR1	SR2	Improv.	U	SR1	SR2	Improv.
EDF-RRJM	7.9	0.3593	0.4052	12.78%	7.9	0.2674	0.3511	31.30%
EDF-MLD-Fair	7.2	0.8204	0.9381	14.35%	7.6	0.6630	0.8152	22.96%
EDF-MLD-WM	7.2	0.8204	0.9381	14.35%	7.6	0.6630	0.8152	22.96%
EDF-MLD-DMIN	7.9	0.3593	0.4141	15.25%	7.6	0.6630	0.8152	22.96%
EDF-MLD-U	7.9	0.3593	0.4052	12.78%	7.6	0.6630	0.8152	22.96%

Table 7: Best improvement, in %, of the success ratio of each algorithm (SR2) w.r.t. partitioned EDF (SR1) vs U for 8 processors

With 8 processors, we present in table 7 a comparative table of the maximum percentage of success ratio improvement. We also provide the value of processor utilization (U) for which this percentage is reached.
If with First-Fit the performance of heuristics is similar with 4 or 8 processors, with Worst-Fit, at very high utilization, the job partitioning EDF-RRJM reveals its potential. When a portioned partitioning approach is limited by the time granularity, EDF-RRJM can always create from a task up to m subtasks of period multiplied by m. For example, suppose that a task $\tau_i(C_i, T_i, D_i) = (1, 2, 2)$ cannot be partitioned and the time granularity is 1. All EDF-MLD-* approaches fail to split this task while EDF-RRJM can create $p \leq m$ subtasks $\tau_i(C_i, pT_i, D_i) = \tau_i(1, 2p, 2)$ and potentially succeed in scheduling the task set. Consequently, it seems that a portioned partitioning cannot always take advantage of an increase in the number of processors while the EDF-RRJM semi-partitioning approach is able to take advantage of it. For very high density and discrete time granularity, EDF-RRJM reaches an improvement of 31.3% compared to partitioned-EDF which represents a gain of about 19.3% compared to EDF-MLD-WM.

6.2.2. Density of migrations

Figure 5 shows the results of simulations based on the density of migrations. Since a migration occurs only when the semi-partitioned technique is used, the results show no migration with low task sets utilization. Our graphs focus on the same range of utilization [3.0;4.0] and [7.0;8.0] respectively with 4 and 8 processors.
In order to obtain representative graphs, we compute the density of migration only for feasible task sets with all the semi-partitioned algorithms. *EDF-RRJM* leads to one migration per job release, whereas *portioned partitioning* produces a number of migrations at most

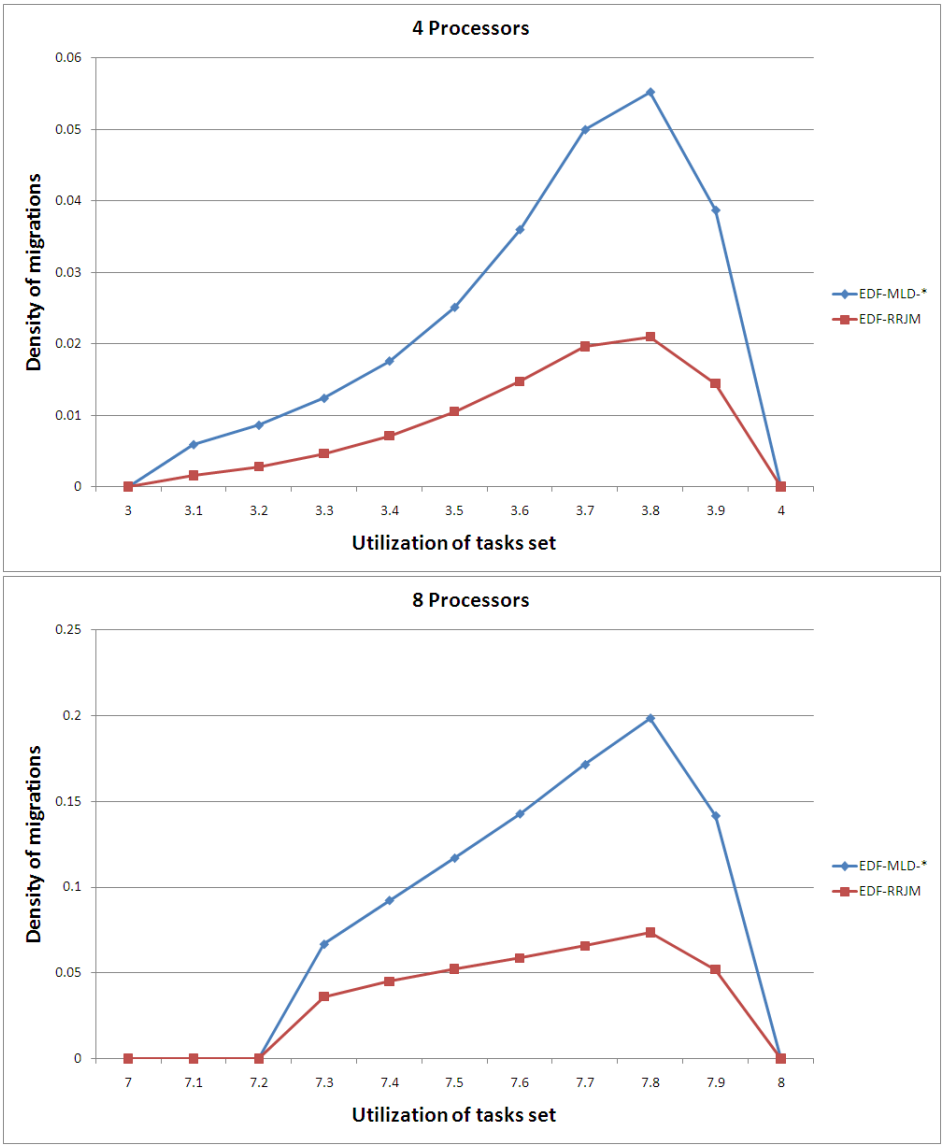

Figure 5: Density of migrations according to utilization of task sets

equal to the number of portions during the period of a task. The density of migration for EDF-RRJM is on average respectively 37% (respectively 43%) of the density of migration for EDF-MLD-* algorithms for 4 (respectively 8) processors. Hence, the average number of migrations obtained with EDF-MLD-* algorithms is 2.69 (respectively 2.32) times the number of migrations of EDF-RRJM for 4 (respectively 8) processors.

7. Conclusion

In this paper we have considered the problem of EDF semi-partitioned scheduling according to two approaches. The first approach, referred to as the portioned semi-partitioning approach, assigns local deadlines to subtasks according to a local deadline assignment heuristic. Based on this local deadline, the maximum acceptable portion of WCET is computed. We have considered several local deadline assignment schemes, according to a fair local deadline assignment and to local deadline assignment based on processor

utilization and the minimum acceptable deadlines of tasks. The migration is done at the local deadline of the job of a task to cancel the release jitter before doing a migration. We show that these heuristics outperform partitioned scheduling by a ratio that can reach 110% for the best heuristic at very high utilization. The second approach we propose, denoted EDF-RRJM, is based on migrations at job boundaries with a Round Robin Job Migration pattern. The solution is easy to implement and results in few migrations. With a First-Fit heuristic, it is outperformed by portioned semi-partitioning heuristics but performs better than classical partitioning scheduling algorithms by a ratio that can reach 44.11%. For a Worst-Fit heuristic with high processor utilization and eight processors, the job semi-partitioning approach performs better than the portioned semi-partitioning approach. In this case, the algorithm based on Round Robin Job Migration outperforms the best portioned semi-partitioned scheduling algorithm by a ratio that can reach 19.3% (under discrete time granularity). Concerning the number of migrations, portioned semi-partitioned schedulings produces at least two times more migrations (on the average) than EDF-RRJM scheduling.

8. References

[1] S. Baruah, N. Fisher, The Partitioned Dynamic-priority Scheduling of Sporadic Task Systems, Vol. Vol. 36, issue 3, Real-Time Systems: The International Journal of Time-Critical Computing., 2007.

[2] J. M. Lopez, J. Diaz, D. F. Garcia, Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems, Vol. pp. 39-68, Real-Time Systems journal, 2004.

[3] S. Kato, N. Yamasaki, Y. Ishikawa, Semi-partitioned scheduling of sporadic task systems on multiprocessors, in: ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems, IEEE Computer Society, Washington, DC, USA, 2009, pp. 249–258.

[4] S. Baruah, J. Gehrke, G. Plaxton, Fast scheduling of periodic tasks on multiple resources, in: Proceedings of the International Parallel Processing Symposium, Vol. pp 280-288, Santa Barbara, California. IEEE Computer Society Press, 1995.

[5] S. Baruah, N. Cohen, G. Plaxton, D. Varvel, Proportionate progress: A notion of fairness in resource allocation, Vol. 15(6), pp. 600-625., Algorithmica, 1996.

[6] H. Cho, B. Ravindran, E. Jensen, An Optimal Real-Time Scheduling Algorithm for Multiprocessors, in: IEEE Real-Time Systems Symposium, Vol. pages 101-110, 2006.

[7] A. Bastoni, B. B. Brandenburg, J. H. Anderson, Cache-related preemption and migration delays: Empirical approximation and impact on schedulability, in: 6th International worskshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'10), in conjunction with the 22nd Euromicro Intl Conference on Real-Time Systems Brussels, Belgium, July 7-9, 2010, 2010.

[8] A. Coskun, A. Kahng, T. Rosing, Temperature- and cost-aware design of 3d multiprocessor architectures, in: Digital System Design, Architectures, Methods

and Tools, 2009. DSD '09. 12th Euromicro Conference on, 2009, pp. 183 –190. doi:10.1109/DSD.2009.233.

[9] S. Bertozzi, A. Acquaviva, D. Bertozzi, A. Poggiali, Supporting task migration in multi-processor systems-on-chip: A feasibility study, in: Design, Automation and Test in Europe, 2006. DATE '06. Proceedings, Vol. 1, 2006, pp. 1 –6. doi:10.1109/DATE.2006.243952.

[10] M. Bertogna, Evaluation of existing schedulability tests for global edf, in: ICPPW '09: Proceedings of the 2009 International Conference on Parallel Processing Workshops, IEEE Computer Society, Washington, DC, USA, 2009, pp. 11–18.

[11] S. Funk, S. Baruah, Restricted EDF migration on uniform multiprocessors, Technique Et Science Informatiques 24 (8), pp 917-938. Hermes-Lavoisier, 2005.

[12] S. Baruah, R. Howell, L. Rosier, Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor, Vol. Vol. 2, pp. 301-324, Real-Time Systems, 1990.

[13] M. Bertogna, M. Cirinei, Response-time analysis for globally scheduled symmetric multiprocessor platforms, in: RTSS '07: Proceedings of the 28th IEEE International Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, 2007, pp. 149–160.

[14] S. Baruah, N. Fisher, The Partitioned Multiprocessor Scheduling of Deadline-constrained Sporadic Task Systems, Vol. pp. 918-923, IEEE Transactions on Computers, 2006.

[15] D. Johnson, Fast algorithms for bin packing, Vol. 8(3):272314, Journal of Computer and Systems Science, 1974.

[16] L. George, J. Hermant, A Norm Approach for the Partitioned EDF Scheduling of Sporadic Task Systems, in: Euromicro Conference on Real-Time Systems, ECRTS'09, Vol. Dublin, Ireland, 2009.

[17] I. Lupu, P. Courbin, L. George, J. Goossens, Multi-Criteria Evaluation of Partitioning Schemes for Real-Time Systems, in: In Proc. of IEEE International conference on Emerging Technologies and Factory Automation (ETFA'2010), 2010.

[18] T. Baker, A comparison of global and partitioned EDF schedulability tests for multiprocessors, in: In Proceeding of the International Conference on Real-Time and Network Systems, Vol. Poitiers, France, 2006.

[19] J. Liu, Real-time systems, Upper Saddle River, New Jersay 07458, Prentice Hall, 2000.

[20] S. Baruah, N. Fisher, Partitioned multiprocessor scheduling of sporadic task systems, in: In Proc. 26th IEEE Real-Time Systems Symposium (RTSS'05), 2005.

[21] N. Fisher, T. Baker, S. Baruah, Algorithms for Determining the Demand-Based Load of a Sporadic Task System, in: Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Vol. pp 135-144, IEEE Computer Society Press, 2006.

[22] J. Goossens, S. Funk, S. Baruah, Priority-driven scheduling of periodic task systems on multiprocessors, Vol. 25, Real-Time Systems, Norwell, MA, USA, 2003, pp. 187–205.

[23] T. P. Baker, Multiprocessor edf and deadline monotonic schedulability analysis, in: RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, 2003, p. 120.

[24] T. P. Baker, An analysis of edf schedulability on a multiprocessor, IEEE Trans. Parallel Distrib. Syst. 16 (8) (2005) 760–768.

[25] S. Baruah, Techniques for multiprocessor global schedulability analysis, in: RTSS '07: Proceedings of the 28th IEEE International Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, 2007, pp. 119–128.

[26] T. P. Baker, S. K. Baruah, An analysis of global edf schedulability for arbitrary-deadline sporadic task systems, Vol. 43, Real-Time Syst., Norwell, MA, USA, 2009.

[27] M. Bertogna, M. Cirinei, G. Lipari, Improved schedulability analysis of edf on multiprocessor platforms, in: ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems, IEEE Computer Society, Washington, DC, USA, 2005, pp. 209–218.

[28] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, Implementation of a speedup-optimal global edf schedulability test, in: ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems, IEEE Computer Society, Washington, DC, USA, 2009, pp. 259–268.

[29] J. H. Anderson, V. Bud, U. C. Devi, An edf-based scheduling algorithm for multiprocessor soft real-time systems, in: In Proc. of the 17th Euromicro Conf. on Real-Time Systems, 2005, pp. 199–208.

[30] S. Kato, N. Yamasaki, Semi-partitioning technique for multiprocessor real-time scheduling, in: Proceedings of the WIP Session of the 29th Real-Time Systems Symposium (RTSS), IEEE Computer Society, 2008, p. 4pp.

[31] B. Andersson, E. Tovar, Multiprocessor scheduling with few preemptions, in: RTCSA '06: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, IEEE Computer Society, Washington, DC, USA, 2006, pp. 322–334.

[32] S. Kato, N. Yamasaki, Real-time scheduling with task splitting on multiprocessors, in: RTCSA '07: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, IEEE Computer Society, Washington, DC, USA, 2007, pp. 441–450.

[33] S. Kato, N. Yamasaki, Portioned edf-based scheduling on multiprocessors, in: EMSOFT '08: Proceedings of the 8th ACM international conference on Embedded software, ACM, New York, NY, USA, 2008, pp. 139–148.

[34] S. Kato, N. Yamasaki, Portioned static-priority scheduling on multiprocessors, in: Proceedings of the 22th IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE Computer Society, 2008, pp. 1–12.

[35] S. Kato, N. Yamasaki, Semi-partitioned fixed-priority scheduling on multiprocessors, in: RTAS '09: Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium, IEEE Computer Society, Washington, DC, USA, 2009, pp. 23–32.

[36] N. Guan, M. Stigge, W. Yi, G. Yu, Fixed-priority multiprocessor scheduling with liu and layland's utilization bound, in: Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS '10, 2010, pp. 165–174.

[37] L. C. Liu, W. Layland, Scheduling algorithms for multi-programming in a hard real time environment, Journal of ACM Vol. 20, No 1, pp. 46-61.

[38] B. Andersson, K. Bletsas, S. Baruah, Scheduling Arbitrary-Deadline Sporadic Task Systems on Multiprocessors, in: In Proc. of the IEEE Real-Time Systems Symposium, Vol. pages 385-394, 2008.

[39] B. Andersson, K. Bletsas, Sporadic Multiprocessor Scheduling with Few Preemptions, in: In Proc. of the Euromicro Conference on Real-Time Systems (ECRTS'08), Vol. pages 243-252, 2008.

[40] K. Lakshmanan, R. Rajkumar, P. Lehoczky, Partitioned Fixed-Priority Preemptive Scheduling for Multi-Core Processors, in: In Proc. of the Euromicro Conference on Real-Time Systems (ECRTS'09), 2009.

[41] P. Hladik, A. Deplanche, S. Faucou, Y. Trinquet, Adequacy between AUTOSAR OS specification and real-time scheduling theory, in: In 2nd IEEE Symposium on Industrial Embedded Systems, Special session on Automotive Embedded Systems (SIES'2007), 2007.

[42] L. Bougueroua, L. George, S. Midonnet, Dealing with execution-overruns to improve the temporal robustness of real-time systems scheduled FP and EDF, in: The Second International Conference on Systems (ICONS'07), Sainte-Luce, Martinique, France, April 22 - 28, 2007.

[43] S. Baruah, J. Goossens, The EDF Scheduling of Sporadic Task Systems on Uniform Multiprocessors, Vol. ISSN: 1052-8725, pp 367-374, Real-Time Systems Symposium, 2008.

[44] L. George, J. Hermant, Characterization of the Space of Feasible Worst-Case Execution Times for Earliest-Deadline-First scheduling, Vol. Vol. 6, Num. 11, Journal of Aerospace Computing, Information and Communication (JACIC), American Institute of Aeronautics and Astronautics (AIAA), 2009.

[45] P. Balbastre, I. Ripoll, Schedulability analysis of window-constrained execution time tasks for real-time control, in: Proc. of the 14^{th} Euromicro Conference on Real-Time Systems (ECRTS'02), 2002.

[46] P. Balbastre, I. Ripoll, A. Crespo, Optimal deadline assignment for periodic real-time tasks in dynamic priority systems, in: Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06), Dresden, Germany July 5-7, 2006.

[47] K. Tindell, J. Clark, Holistic schedulability analysis for distributed hard real-time systems, Vol. Vol. 40, Microprocessors and Microprogramming, Euromicro Journal, 1994.

[48] D. Marinca, P. Minet, L. George, Analysis of deadline assignment methods in distributed real-time systems, Vol. Elsevier, Computer Communications, 2004.

[49] K. Tindell, A. Burns, A. J. Wellings, Analysis of hard real-time communications, Real-Time Systems Vol. 9, pp. 147-171.

[50] A. Sahoo, W. Zhao, Partition-based admission control in heterogeneous networks for hard real-time connections, in: Proc. of the 10th International Conference on Parallel and Distributed Computing, 1997.

[51] T. Baker, A comparison of global and partitioned schedulability tests for multiprocessors, in: Technical Report TR-051101, Vol. FSU Computer Science, 2005.