Job Partitioning Strategies for Multiprocessor Scheduling of Real-Time Periodic Tasks with Restricted Migrations

Joël Goossens  
Université libre de Bruxelles, Belgium  
joel.goossens@ulb.ac.be

Pascal Richard  
LIAS/Ensma & Université de Poitiers, France  
pascal.richard@univ-poitiers.fr

Markus Lindström  
Université libre de Bruxelles, Belgium  
mlindstr@ulb.ac.be

Irina Iulia Lupu  
Université libre de Bruxelles, Belgium  
irinlupu@ulb.ac.be

Frédéric Ridouard  
LIAS/Ensma, France  
fréderic.ridouard@ensma.fr

ABSTRACT
In this work, we consider the scheduling of real-time periodic tasks on multiprocessors under a restricted-migration constraint (tasks are allowed to migrate whereas jobs are not). We first propose a technique, (offline) job partitioning, which statically assigns jobs to processors before execution. We show that (offline) job partitioning strictly dominates both partitioned and restricted-migration scheduling. We then consider r-RM, a restricted-migration variant of global RM, we design a sufficient schedulability test and we provide a speedup factor for the scheduler.

Categories and Subject Descriptors
C.3 [Real-time and embedded systems]:

General Terms
Theory

1. INTRODUCTION
Uniprocessor scheduling has been widely studied over the last few decades. However, a single processor can no longer satisfy today’s computational demands, as the miniaturization of integrated circuits reaches its physical limits [38]. Thus, a valid solution to supply enough resources is the use of multiprocessor platforms. Specifically, multicore CPUs offer reduced inter-core communication costs compared to traditional multiprocessor platforms. This creates the potential for significantly more efficient scheduling strategies. Improvements of migration strategies, or core-switching, enable enhanced and fine-grained load-balancing as well as thermal management of cores. Furthermore, recent advances have been made in ensuring predictability and bounded deterministic migration delays for hard real-time systems implemented upon multicore platforms [36].

Multiprocessor scheduling algorithms can be classified according to two fundamental decisions that must be taken for every pending job in the system: the processor the job must be allocated to, and the order in which the jobs must be executed with respect to jobs of other tasks [12,14]. Job ordering is usually done using a priority-driven scheduling policy at runtime, whereas the allocation problem can be either solved online and/or offline. Next, we briefly describe the three main priority schemes and migration strategies.

Regarding the priority criterion, the following categories have been identified: fixed-task priority, fixed-job priority and unrestricted dynamic priority. A Fixed-Task Priority (FTP) rule assigns a unique priority to each task, and all jobs generated by a task inherit the priority of that task (e.g. Rate Monotonic [30], Deadline Monotonic [6]). A Fixed-Job Priority (FJP) policy assigns a fixed and unique priority to each job, and two jobs of the same task may have distinct priorities (e.g. Earliest Deadline First [30]). An Unrestricted Dynamic Priority (UDP) rule places no restrictions on the priorities that may be assigned to jobs, and a job may have different priority levels during its lifetime (e.g. Least Laxity First [15]).

The following categories are usually considered with regards to the migration criterion [12,14]. (i) No migration: each task is allocated to a processor and no migration is allowed. (ii) Task-level migration: the jobs of a task may execute on different processors. However, a given job may only execute on a single processor. (iii) Job-level migration: any job can migrate during its lifetime. According to Partitioned Scheduling (PS), the set of tasks is statically (i.e. by an offline algorithm) partitioned into a number of disjoint subsets. This number is bounded by the number of processors of the platform. Each subset is then assigned to a processor and a uniprocessor scheduling method is applied locally. No migration is allowed. According to Global Scheduling (GS), a task may migrate from one processor to another without any constraints in order to meet its deadlines. In practice, job migrations can cause a prohibitive overhead. Restricted-migration Scheduling (RS) allows, like
1. RELATED WORK

Restricted-migration scheduling is an intermediate model that allows tasks to migrate only at job boundaries and seems to be an interesting compromise between global and partitioned scheduling. Our work concerns job partitioning strategies to enforce migrations at job boundaries (i.e. restricted-migration model). Such a model can be viewed as a generalization of (i) the task-level migration model and (ii) the restricted-migration model.

2. SYSTEM MODEL

The task model considered in this paper is the periodic task model: each task $\tau_i$ in a task system $\tau$ is characterized by the tuple $(e_i, p_i)$ where $e_i$ is the execution demand and $p_i$ is the exact inter-arrival time between two successive activations of the task, also called period. The task is instantiated an infinite number of times. Therefore, it generates an infinite number of jobs. The execution demand of a job has to be satisfied before the next activation of the same task (implicit deadline model). In this paper, $\tau_i$ denotes the job of task $\tau_i$. An active job is a job that has not completed its execution. $P$ designates the least common multiple of all the task periods in the system: $P \overset{\text{def}}{=} \text{lcm}\{p_1, \ldots, p_n\}$.

Each task is characterized by a utilization $u_i \overset{\text{def}}{=} \frac{e_i}{P}$. If $u_i$ is less than or equal to one half, we say that $\tau_i$ is a light task. Otherwise, it is a heavy task. The total utilization of the system $\tau$ is denoted by $U_{\text{total}}(\tau) \overset{\text{def}}{=} \sum_{i=1}^{n} u_i$. By $U_{\text{max}}(\tau) \overset{\text{def}}{=} \max_{i=1}^{n}(u_i)$, we understand the maximum individual task utilization of the system.

The considered multiprocessor platform contains $m$ identical processors ($\tau_j$ denotes the $j$th processor in the platform). Each processor has unit capacity, meaning that the total utilization of tasks running on that processor is at most 1 (which yields a necessary feasibility condition). Usually, a task is allowed to migrate in order to increase the chances to complete by its deadline. However, it is very difficult to estimate the real cost of a migration (there are many parameters that must be accounted for, such as the moment when the task is migrated during execution, the processor to which the task is migrated to, etc.) and often this cost is modeled within the worst-case execution time of every job (WCET).

A task set is feasible if there exists a schedule that ensures it never misses a deadline. A task is referred to as schedulable with regards to a given scheduling algorithm $A$, if all the jobs complete by their deadline in the schedule computed by $A$ at runtime. We say that a scheduling algorithm $A$ is work-conserving if the processors are never idle as long as there are active jobs awaiting execution.

A schedulability test for a scheduling policy is sustainable [7] if any system deemed schedulable by the schedulability test remains schedulable when the parameters of one or more individual jobs are changed in any, some, or all of the following ways: (i) decreased execution requirements, (ii) later arrival times, (iii) smaller jitter, and/or (iv) larger relative deadlines. In the framework of this research, we are focusing on sustainable schedulers with respect to the execution requirement parameters (i.e. also called predictability [22]). Let us consider the sets of jobs $J$ and $J'$ which differ only with regards to their execution times in the manner that the jobs in $J$ have execution times less than or equal to the execution times of the corresponding jobs in $J'$. A scheduling algorithm $A$ is sustainable with respect to the execution requirements if, when applied independently on $J$ and $J'$, all jobs in $J$ finish execution before or at the same time as the corresponding jobs in $J'$.

3. RELATED WORK

Global scheduling, the migration of a task from one processor to another, but only at job boundaries: once a job starts its execution on a CPU, it is then mandatory to complete upon that same CPU. In this case, job migration is forbidden whereas task migration is allowed.

The restricted-migration model is an intermediate model between global scheduling and partitioned scheduling models. Partitioned scheduling is clearly a particular case of restricted-migration scheduling and, if preemptions are not allowed, restricted-migration and global scheduling are equivalent [12]. It is known in the literature that if job-level migrations are forbidden, then the maximum utilization bound of the multiprocessor platform is $(m+1)/2$, where $m$ is the number of identical processors. Thus, one half of the total processor capacity may be wasted if migrations are forbidden or only allowed at job boundaries. Hybrid approaches have been proposed to combine partitioned and global approaches in order to cope with such poor worst-case utilization of the platform processing capacity by introducing as few job-level migrations as possible [14] (e.g. semi-partitioned approaches).

This research. In this work, we consider the restricted-migration platform model. Our objective is to provide some insight on migration strategies, both in online and offline settings. On one hand, we consider the case of offline job partitioning, where the jobs of all tasks are statically assigned to processors before execution. On the other hand, we consider online job partitioning of fixed-task priority (FTP) systems: a given job is allocated to an arbitrary processor at runtime according to the result of an admission test (i.e. uniprocessor schedulability test). The contributions of this paper are summarized in Table 1 following the categorization presented in [12].

<table>
<thead>
<tr>
<th>Key</th>
<th>Online Job Partitioning</th>
<th>Static Job Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td>$U_{\text{max}}(\tau) \leq \frac{m+1}{2}$</td>
<td>$U_{\text{max}}(\tau) \leq m$</td>
</tr>
<tr>
<td>FJP</td>
<td>$U_{\text{max}}(\tau) \leq \frac{m+1}{2}$</td>
<td>$U_{\text{max}}(\tau) \leq m$</td>
</tr>
<tr>
<td>UDP</td>
<td>$U_{\text{max}}(\tau) \leq \frac{m+1}{2}$</td>
<td>$U_{\text{max}}(\tau) \leq m$</td>
</tr>
</tbody>
</table>

Table 1: Our contributions
semi-partitioned systems that perform job partitioning after a task partitioning stage (using bin-packing techniques).

Of all migration strategies for job-level and priority-driven scheduling algorithms, EDF-based scheduling algorithms are the most studied for multiprocessor systems. Although EDF is not an optimal scheduling algorithm for scheduling real-time tasks upon m identical processors, it has the least run-time complexity among fixed job priority scheduling algorithms and its simple behavior allows definition of efficient schedulability tests. The performance of scheduling algorithms are compared through quantitative metrics such as the maximum utilization bound (e.g. based on the number of processors m, the total task utilization $U_{\text{sum}}(\tau)$ and the maximum task utilization $U_{\text{max}}(\tau)$) and the resource augmentation ratio (e.g. speedup factor for scheduling a feasible task set).

For the maximum utilization bound metric, it is known that no priority-driven scheduling algorithm has a schedulable utilization bound greater than $(m+1)/2$ [37]. Partitioning algorithms are usually based on bin-packing heuristics. Among them, algorithms which fail to allocate a task only when there is no processor in the system which could hold it are called reasonable allocation algorithms [31]. For partitioned EDF, Lopez et al. [31] proved that any reasonable allocation algorithm matches that best possible utilization bound of $(m+1)/2$ (i.e. offline algorithms such as any decreasing fit, and the first-fit and best-fit online algorithms).

In the restricted-migration setting, the maximum utilization bound of EDF is $m - (m-1)U_{\text{max}}(\tau) \leq U_{\text{sum}}(\tau) \leq \frac{m}{m+1}$ [8]. For partitioned RM, if the Liu and Layland schedulability test is used as an acceptance test, the maximum utilization bound is $(\sqrt{2} - 1)$ [32,33] and the first-fit algorithm matches the bound in [34]. However, under the assumption that the Rbound schedulability test [29] is used instead of Liu and Layland’s, Andersson and Jonsson [5] define a next-fit based algorithm that achieves the maximum utilization bound of $m/2$. Clearly, these results state that approximately 50% of the processing capacity can be wasted when EDF is considered. This is unfortunate because priority-driven scheduling algorithms entail lower scheduling and task migration overheads when compared to optimal schedulers (e.g. PFAIR algorithms).

The second quantitative metric used to compare scheduling algorithms is the resource augmentation ratio ([35]) which defines the amount of extra processor speed that must be given to an online algorithm in order to schedule a feasible task set upon a unit-speed processor platform. Such a speedup factor can be easily defined from a Maximum Utilization Bound (MUB) for a given scheduling algorithm $A$, denoted $\text{MUB}_A$ hereafter. Since every feasible task set $\tau$ satisfies $U_{\text{max}}(\tau) \leq 1$ and $U_{\text{sum}}(\tau) \leq m$, then the speedup factor required for $A$ to meet all deadlines is $s = m/\text{MUB}_A$. For instance, we have already noticed that there exist priority-driven algorithms that guarantee a (maximum) utilization bound of $(m+1)/2$. Thus, the required speedup factor for scheduling a feasible task set is $2m/(m+1) = 2 - \frac{1}{m+1}$. Several results on resource augmentation analysis of EDF and RM are summarized in [19] both in global and partitioned scheduling settings. Nevertheless, positive but complex algorithms have been defined with a parametric speedup factor (i.e. Polynomial Time Approximation Schemes, or PTAS).

Based on the principles of a PTAS defined for solving a makespan scheduling problem [23], Baruah defined a PTAS for EDF with the following performance guarantee: the EDF$_{\varepsilon}$ algorithm (i.e. EDF with an accuracy parameter $\varepsilon$) computes a partition on $m$ processors, each of speed $(1 + \varepsilon)$, in polynomial time with respect to the size of the representation of a task set $\tau$ (subject to some assumptions on task parameters). To the best of our knowledge, no resource augmentation analysis result is known for online scheduling algorithms with restricted-migration.

Only few results concern multiprocessor restricted-migration scheduling. We briefly summarize these results in the online, offline and semi-partitioned settings.

**Online job partitioning.**

Online job partitioning is as difficult as online task partitioning (thus NP-hard in the strong sense). Furthermore, there is no optimal online algorithm for partitioning jobs with two distinct deadlines [24]. Clearly, job partitioning is related to solving an online bin-packing problem under the constraint that assigned jobs are feasible on the selected processor. It is important to notice that an online bin-packing algorithm leads to anomalies when periods or deadlines are increased or when a processing requirement decreases for some tasks (see [3] for numerous detailed examples or [13] in the online bin-packing context). Nevertheless, some practical solutions, detailed hereafter, have been proposed for the online scheduling of tasks with restricted-migration.

For $m$ identical (unit-capacity) processors, restricted-migration scheduling algorithms based on EDF have been proposed [4,9]. These algorithms use EDF for scheduling jobs on each processor and basically only differ in the new job admission test formulations, since Baruah et al. consider periodic tasks whereas Andersson et al. consider aperiodic tasks [4]. However, both admission tests in these algorithms are based on the ratio of execution requirements over periods or relative deadlines computed for active jobs (both are in practice linked to the notion of *synthetic utilization* that has been formally studied in [1]). In [4], the admission test checks whether the synthetic utilization is less than or equal to 0.31$m$, whereas in [9], the admission checks whether current processor utilization is less than or equal to one (i.e. all jobs of the considered task are admitted). Basically, these algorithms update, for each processor, the utilization of currently assigned jobs at time $t$. If a processor has sufficient available utilization, the job is assigned to it and the corresponding utilization is removed from the considered processor utilisation; when the job completes, then its processor is added to the available processor utilization. $r$-EDF is an online algorithm that is valid for scheduling light tasks with utilization less than or equal to one-half. The r-Prid algorithm is a variant of $r$-EDF that statically allocates heavy tasks with a utilization of strictly more than one-half to some dedicated processors, whereas light tasks are scheduled online using $r$-EDF. The distinction between light and heavy tasks is required for the design of utilization-based schedulability tests. Such an approach has also been successfully extended for uniform multiprocessor platforms [20]. In [17], Fauthauet al. consider the multiprocessor scheduling of fixed-task priority periodic tasks (priority ordering is done with the Deadline Monotonic algorithm). A sustainable (with regards to execution requirements) laxity-based allocation algorithm is presented for solving the online assignment of jobs to the processors. The proposed algorithm is compared to the Deadline Monotonic algorithm both in
global and restricted-migration settings. Finally, it has been shown that no resource augmentation guarantee can be obtained if synthetic utilization is considered as a job admission test [18].

**Offline job partitioning.**

In the offline setting, deciding restricted-migration feasibility is known to be intractable. Indeed, the problem of deciding whether a given task system $\tau$ can be scheduled upon $m$ unit capacity processors by any restricted-migration algorithm has been shown to be NP-hard in the strong sense [8, 9].

In [10], such an offline algorithm is presented (JobAssign). It copes with a finite set of jobs with release dates and relative deadlines. Jobs are assigned to processors one by one in increasing order of their relative deadlines until they satisfy a uniprocessor feasibility test. A load-based feasibility test is defined and its resource augmentation analysis leads to an upper bound of the speedup factor equal to $4 - 1/m$ against an optimal algorithm, where $m$ is the number of processors.

**Semi-partitioning.**

Semi-partitioning is an example of semi-online scheduling algorithms (i.e. the scheduler has some partial information about the forthcoming jobs, see [2] for details). The first stage is performed offline and consists in task partitioning using a bin-packing heuristic, and the second stage is performed online and tries to allocate to the processors the remaining jobs with restricted-migration [16, 27] or the remaining portions of jobs if job-level migration is allowed [21, 26]. In the latter case, it is assumed that jobs can be ported, and for all these portions a release time and a deadline must be defined. In [2], an EDF-based semi-partitioned algorithm, called EDF-fm, is presented for scheduling soft real-time tasks. Each task is allocated to two processors at most. When a task is allocated to two processors, migrations are only allowed at job boundaries. EDF-fm allows to define a maximum task tardiness bound (i.e. the maximum delay between a job completion and its deadline at runtime).

### 4. STATIC VS. ONLINE JOB PARTITIONING

In this section we will present an offline job partitioning technique based on periodic functions. We then compare such offline job assignments to online algorithms such as those used in r-EDF or r-Prid for allocating jobs to the processors.

Offline and online job partitioning differ in the instant at which job allocations are made. As in offline partitioned scheduling, offline job partitioning assumes that an offline algorithm computes a data structure (i.e. a mapping table) for storing which processor is assigned to every job. Such a data structure is then used online by the scheduler implemented in the real-time kernel to allocate a given job according to the mapping table. Then, a classic online scheduler is used to schedule jobs on every processor. If job partitioning is performed online, then such a data structure is not required as allocation decisions are made whenever jobs arrive in the system.

Since we consider periodic tasks, every task generates an infinite collection of jobs. We define an offline job partitioning by an $n$-tuple $s \overset{\text{def}}{=} (s_1, s_2, \ldots, s_n)$. According to this job partitioning $s$, the jobs in the system are statically assigned to processors. More precisely, each task $\tau_i$ is characterized by a periodic job assignment sequence $s_i$ of length $k_i > 0$ (the periodicity starts with the first job). The job assignment sequence $\tau_i$ is $s_i \overset{\text{def}}{=} (a_1^i, a_2^i, \ldots, a_{k_i}^i)$ with the following interpretation: job $\tau_{i,j}$ is assigned to processor $\pi_{a_j (j - 1 \mod k_i) + 1}$

Notice that the particular case where $k_i = 1$ ($\forall i$) corresponds to partitioned scheduling. On the other hand, if $\exists \tau_i \in \tau$ such that $k_i > 1$ and $a_j^i \neq a_j^i$ ($j \neq \ell$), $j, \ell \in \{1, \ldots, k_i\}$, we have at least one that task migrates between two or several processors.

In Figure 1, a schedule corresponding to an offline job partitioning scheme is presented for a task system $\tau$ composed of $\tau_1 = (2, 3)$ ($s_1 = (1)$), $\tau_2 = (3, 4)$ ($s_2 = (2))$ and $\tau_3 = (2, 6)$ ($s_3 = (2, 1)$). We can observe that $\tau_2$ is executing only on the first processor and $\tau_2$ on the second one, whereas $\tau_1$ shares its execution between both processors.

#### 4.1 Sustainability

After offline job partitioning, the scheduling problem reduces to $m$ uniprocessor problems. Thus, offline job partitioning scheduling is sustainable (with respect to the execution requirements) if the uniprocessor schedulers are sustainable (with respect to the execution requirements). This is the case for all priority-driven uniprocessor schedulability policies [24].

This is a major property in favor of offline job partitioning and against online job partitioning. Indeed, Ha and Liu showed that restricted-migration scheduling (i.e. jobs can be preempted but not migrated) are non-sustainable with respect to the execution requirements (see [22], Figure 1 for details). As a consequence, the schedulability test corresponding to an online job partitioning scheduling algorithm must be sustainable.

#### 4.2 Number of processors per migrating task

In order to reduce the search space for a feasible schedule for a given restricted-migration system, we thought of restricting the migration of a task on a specific number of processors. For instance, remember that EDF-fm [2] considers tasks allowed to migrate upon at most two processors at job boundaries. We next show that such an assumption is too restrictive to define the best possible migration strategies in the restricted-migration setting.

We will now show that there are systems which need to allow a task to migrate upon all the processors of the platform in order to meet its deadlines. Consequently, no restriction can be imposed on the number of processors per migrating task. For instance, Counter-example 1 illustrates this case.

**Counter-example 1.** We consider $m$ processors and a
task system $\tau = \{\tau_1, \ldots, \tau_m, m+1\}$ with $p_1 = \cdots = p_m = 12$, $e_1 = \cdots = e_m = 10$, $p_{m+1} = 4$, $e_{m+1} = 2$. For a feasible job assignment, each of the first $m$ tasks has to be assigned on an individual processor and the $(m+1)\text{th}$ must be allowed to migrate over the entire platform. For example: $s_i = (i), 1 \leq i \leq m$ and $s_{m+1} = (1, \ldots, m)$. Figure 2 presents the corresponding schedule.

<table>
<thead>
<tr>
<th>time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_1$</td>
<td>$\tau_{m+1,1}$</td>
<td>$\tau_{1,1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\pi_2$</td>
<td>$\tau_{2,1}$</td>
<td>$\tau_{m+1,2}$</td>
<td>$\tau_{2,1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\tau_{m,1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\pi_m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\tau_{m+1,1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Migration example

4.3 Work-conservative job partitioning & dominance properties

In Figure 1, we can observe that offline job partitioning leads to non work-conserving scheduling algorithms, regardless of the online priority rules used on every processor. According to the considered partitioning, at time instant 2, $\tau_{1,1}$ is ready and not running even though the first processor is available. We can conclude that globally, offline job partitioning is not work-conserving. However, after offline job partitioning, subsets of jobs are statically allocated to a given processor and are scheduled according to a uniprocessor policy. Therefore, offline job partitioning can be stated to be locally work-conserving on each processor.

We have to mention that online job partitioning always assigns an active job if there is a processor that admits it according to a uniprocessor feasibility test. Thus, online job partitioning leads to a work-conserving scheduling policy if the uniprocessor scheduling algorithm itself is work-conserving.

In [9], it was shown that online restricted-migration scheduling (RS) and partitioned scheduling (PS) are incomparable (there are task systems schedulable by RS and not by PS, and conversely). In the following, we will prove that offline job partitioning dominates both RS and PS.

**Theorem 1.** Offline job partitioning strictly dominates online job partitioning.

**Proof.** This statement is based on the fact that every online restricted-migration schedule can be modeled by periodic functions modeling job assignments in an offline job partitioning schedule. The dominance property is strict since RS is work-conserving and offline job partitioning is not.

The domination is strict since the opposite property is not satisfied as will be proved by Counter-example 2. This example was used in [12] to prove that there are task systems infeasible by RS, but feasible by PS. Let us take an example of system that is schedulable using an offline job partitioning and that will not be schedulable using online job partitioning.

**Counter-example 2.** (System $D$ of [12]). We consider the system $\tau = \{(1, 3), (2, 3), (3, 6), (4, 6), (5, 6)\}$ ($m = 2$) in Figure 3.

![Figure 3: No online job partitioning leads to a feasible schedule [12]](image)

In time interval $[0, 6]$, all jobs finish executing before their deadlines regardless of whether they are prioritized relatively to each other. Since according to an online job partitioning algorithm, a job occupies the first available processor upon arrival, at time instant 6, the second job of $\tau_2$ will migrate to the second processor. The second job of $\tau_2$ will miss its deadline since it has only 5 time units available for execution until instant 14, but 6 units of execution demand.

The task system is schedulable with offline job partitioning since jobs may be partitioned among the processors in the platform as follows: the jobs of tasks $\tau_1$ and $\tau_2$ are assigned to $\pi_1$ and the jobs of $\tau_3$ are assigned to $\pi_2$. This is equivalent to partitioning tasks among the processors in the platform.

**Theorem 2.** Offline job partitioning strictly dominates task partitioning.

**Proof.** Since PS statically assigns tasks to processors, the jobs of each task execute on a unique processor. An equivalent offline job partitioning can be built with the same job assignment on processors as for the PS algorithm. The domination is strict since the opposite property is not satisfied as shown in the Counter-example 1.

Next, we present a simple counter-example showing that ties cannot be broken arbitrarily while using EDF as a local scheduler on each processor in the restricted-migration setting.

**Counter-example 3.** Consider two processors $\pi_1$ and $\pi_2$, and the following task set: $\tau = \{(1, 3), (2, 3), (3, 6), (4, 6), (5, 6)\}$. Consider the EDF-restricted migration schedule presented in Figure 4 in which EDF tiebreaker will take several decisions:

- at time 0, $\tau_3$ has the highest priority and the other tasks have the same deadlines, and according to EDF, ties are broken arbitrarily. Let us assume that $\tau_2$ is selected by the EDF tiebreaker.
- at time 2, the first job of $\tau_3$ completes and let us assume that the EDF tiebreaker selects $\tau_4$ for starting its execution on $\pi_1$.
- at time 3, the second job of $\tau_3$ is released and $\tau_1, \tau_2$ are ready. These 3 jobs are ready and have the same deadline at time 6. Assume that the EDF tiebreaker allocated $\tau_2$ to the processor $\pi_1$ and the two other jobs to $\pi_2$, as depicted in Figure 4. Such a tie breaking leads $\tau_3$ to be preempted at time 3 and resumed at time 6 on the same processor, and thus it misses its deadline.

The schedule presented in Figure 5 only differs by tie breaking decisions taken at time 3. As depicted in that figure, all deadlines are met.
62
\[ \tau \]
4
3
2
\[ \text{time} \]
\begin{array}{cccccccc}
\tau & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_1 & 73,1 & 74,1 & 73,2 & 72,1 & 71,1 & 72,1 & 72,1 \\
\pi_2 & 73,1 & 74,1 & 71,1 & 72,1 & 71,2 & 72,1 & 72,1 \\
\end{array}

Figure 4: edf-restricted migration with tie breaking leading to an infeasible schedule

4.4 Worst-case achievable utilization

A common way to characterize the performance of scheduling algorithms is to use the notion of utilization bound. The utilization bound of a real-time scheduling algorithm \( A \) on a given platform is the maximum number \( U_B \) such that for every task system \( \tau \) with total utilization \( U_{\text{sum}}(\tau) \leq U_B \) all deadlines are met when executing. It has been shown that only global scheduling with dynamic priority can lead to a worst-case utilization greater than \((m+1)/2\) [14].

Thus, online and offline job partitioning strategies will lead to the same maximum utilization bound when identical multiprocessor platforms composed by \( m \) processors are considered. To see that, consider the following example (which was also presented in [9] for the online case that is still valid even if offline job partitioning is allowed): consider \((m+1)\) identical tasks with an individual utilization of \( \frac{1}{m+1} \). Since all the tasks are identical, their periods are equal to \( P \) and there is one job in the time interval \([0,P]\) for each task \( \tau_i \in \tau \). There is no possible job partitioning without job migration for the given task set. As \( \varepsilon \) approaches 0, the total utilization of such a task system approaches \((m+1)/2\).

5. OFFLINE JOB PARTITIONING

5.1 The allocation function is periodic

A job partition is, for each task \( \tau_i \), characterized by the vector \( s_i = (a_1^i, a_2^i, \ldots, a_{n_i}^i) \) with the interpretation that job \( \tau_{i,j} \) is assigned to processor \( \pi_{(j-1) \mod k_i} \) and thus far no restriction has been placed upon the value of \( k_i \). We will now show that, without loss of generality, we can limit ourselves to considering the following case:

\[ k_i = \frac{P}{p_i}, \forall i \in \{1, \ldots, n\}. \]

where \( P \equiv \text{lcm}(p_1, \ldots, p_n) \). More formally:

Theorem 3. For any task system \( \tau \), if there exists a feasible job partition \( s \), there must exist a feasible job partition \( s' = (s'_1, s'_2, \ldots, s'_n) \) such that \( k'_i \equiv \#s'_i = \frac{P}{p_i} \).

Proof. The principle of the proof is the following: based on a feasible job partition where \( \exists k \in \{1, \ldots, n\} \) such that \( k_i \neq \frac{P}{p_i} \), we can build another feasible job partition such that \( k_i = \frac{P}{p_i}, \forall i \in \{1, \ldots, n\} \).

Let us consider the \( A \)-schedule of \( \tau \) such that for at least one task \( \tau_i \in \tau \), the inequality \( k_i \neq \frac{P}{p_i} \) holds. Since the system is \( A \)-schedulable, this means that all the jobs in the time interval \([0,P]\) finish executing before their deadlines. We know that \( P \) is the \( \text{lcm}(p_1, \ldots, p_n) \) by definition, and since we assume tasks have implicit deadlines, the execution demand of time interval \([0,P]\) is satisfied before \( P \). At instant \( P \), all tasks reactivate simultaneously. Consequently, in every time interval \([\ell P, (\ell + 1)P] \) \( (\ell \geq 1) \), the same partition as the one in \([0,P]\) can be applied while maintaining the schedulability of the system.

5.2 Feasible Job Partition Search Algorithm (FJPS)

In order to find a job partition that is feasible, we have to consider all the possible partitions of the set of jobs on the processors. This means that for a platform containing \( m \) processors, we have to consider all the possible job partitions for each of the following cases: 1 processor is used, 2 processors are used, \( \ldots, \) \( m \) processors are used.

As shown in Theorem 3, we can consider without loss of generality only the jobs in the time interval \([0,P]\). In the following \( K \equiv \sum_{i=1}^{m} k_i \) (i.e. the number of jobs that have to be assigned to processors in the time interval \([0,P]\)).

In the following, a \( v \)-partition \( (v \in \{1, \ldots, m\}) \) represents a partition of the set of \( K \) jobs into \( v \) disjoint subsets. Each subset is assigned to a distinct processor (thus, \( v \) processors are used). The assignment order is not important since all the processors in the platform are identical. Thus, there is no order relation between the considered \( v \) subsets of the given \( v \)-partition. This means that for the set \( \{a, b\} \), \( \{\{a\}, \{b\}\} \) and \( \{\{b\}, \{a\}\} \) denote the same 2-partition. We can also conclude that the set has a single 2-partition. The number of \( v \)-partitions for a given set of \( K \) elements is denoted by the number \( S(K, v) \) and it is called a Stirling number of the second kind:

\[
S(K, v) = \sum_{j=0}^{v} (-1)^{v-j} \binom{v}{j} j^K.
\]

The number of all the partitions of a given set of \( K \) elements into \( 1, 2, \ldots \) and \( m \) subsets is given by a Bell number:

\[
B(K, m) = \sum_{i=1}^{m} S(K, i).
\]

Generating all \( B(K, m) \) partitions can be done using e.g. Hutchinson’s restricted growth string algorithm (Alg.7.2.1.5H in [28]). Orlov has also given a variant of this algorithm that is easier to implement [34]. Each iteration of these algorithms provides a valid partition.

Since EDF is optimal on uniprocessors, a system with offline job partitioning is feasible if and only if the set of jobs given to a processor is EDF-schedulable. Consequently, feasibility of a given partition can be tested by checking EDF-schedulability in the time interval \([0,P]\) for each of the \( m \) processors with the jobs determined by the partition.

In practice, it is sufficient to check feasibility of the only \( S(K, \min(m, K)) \) \( \min(m, K) \)-partitions. Indeed, if \( K < m \),
a straightforward solution exists upon \( K \) processors. Otherwise, if \( K > m \), we can limit ourselves to considering, without loss of generality, solutions with exactly \( m \) partitions (by moving jobs from a solution using \( m' < m \) partitions, it is easy to devise a solution using \( m \) partitions). Algorithms to generate \( e \)-partitions alone are given in both [28] and [34].

FIPS works by generating partitions and running a feasibility test on each of them. If the system is deemed feasible, the algorithm terminates. Otherwise, we proceed to build and then visit the next possible partition (if any, otherwise the algorithm returns a failure).

Note that the algorithm requires exponential time in the worst case, as Stirling numbers of the second kind have exponential growth.

6. ONLINE JOB PARTITION WITH FIXED-TASK PRIORITIES

In [9] an EDF restricted-migration algorithm (r-EDF) is presented. In the following, we will extend this approach to FTP. We will define the r-RM scheduling algorithm: an RS-migration algorithm based on the Rate Monotonic (RM in short where task priorities are inversely proportional to task periods).

6.1 Definition of r-RM

The r-RM algorithm classifies tasks into two disjoint subsets: heavy tasks and light tasks. Heavy tasks have an individual utilization \( u > 0.5 \). For light tasks, the individual utilization \( u < 0.5 \).

The two task subsets are handled differently by the r-RM algorithm:

1. firstly, each heavy task is assigned to an individual processor;
2. the remaining light tasks are scheduled on the remaining processors using the global RM priority assignment.

The r-RM algorithm is based on the same data structures as r-EDF [9]:

- a local scheduling queue \( Q_j \) which stores, at each time instant \( t \), the priorities (according to RM) of the active jobs assigned to processor \( \pi_j \),
- a current utilization \( U_j \) which is a measure of the load that has been assigned on processor \( \pi_j \) thus far,
- a decrement list \( L_j \) of ordered pairs of real numbers with each such ordered pair \( (t, U) \in L_j \) indicating that the current utilization should be incremented by an amount \( U \) at time-instant \( t \). At the instant \( t_0 \) such that ordered pair \( (t_0, U) \in L_j \), the current utilization \( U_j \) is decremented by amount \( U \) and \( (t_0, U) \) is removed from \( L_j \); hence, each \( L_j \) at all the instants \( t_1 \) only contains entries with the first coordinate \( > t_1 \).

The assignment procedure is based on the one used by r-EDF [9]. At time instant \( t_{\text{cur}} \) prior to any job assignment, \( U_j \) (\( \forall 1 \leq j \leq m \)) will have been decremented as demanded by the corresponding \( L_j \). In particular, if \( (t_{\text{cur}}, U) \in L_j \), \( U_j \) will firstly be decremented by an amount of \( U \) before any assignment is done. Then, the newly-arrived job \((e, p)\) is assigned. The job assignment procedure then:

1. Verifies the nature of the task that generated the job:
   (a) if the task is heavy, the job is assigned to the statically chosen processor \( \pi_j \) (during the partitioning phase of heavy tasks);
   (b) if the task is light, the job is assigned to some processor \( \pi_j \) (among those available for light tasks) that can accommodate it, i.e. some processor \( \pi_j \) with current utilization \( U_j \) satisfying
      \[
      U_j \leq U_B - \frac{e}{p},
      \]
   where \( U_B \) represents a uniprocessor (utilization) bound for the RM scheduler as the one expressed in [11] with \( U_B = \prod_{i=1}^{n} (1 + u_i) \) or the well-known Liu and Layland’s maximum utilization bound [30] defined as \( U_B = n(\sqrt{2} - 1) \geq \ln(2) \).

2. Increments \( U_j \) by \( \frac{e}{p} \) to reflect the fact that the load of processor \( \pi_j \) has been increased by an amount equal to \( \frac{e}{p} \).

3. Records the fact that \( U_j \) is to be incremented by an amount equal to \( \frac{e}{p} \) at time instant \( t_{\text{cur}} \) since the newly assigned job will contribute to the load on processor \( \pi_j \) only during the time interval \( [t_{\text{cur}}, t_{\text{cur}} + p) \). This is done as follows:
   - if there already is an ordered pair \( (t_{\text{cur}} + p, U) \) in \( L_j \), then replace it by the ordered pair \( (t_{\text{cur}} + p, U + \frac{e}{p}) \);
   - else add the ordered pair \( (t_{\text{cur}} + p, \frac{e}{p}) \) to \( L_j \).

6.2 Utilization Bound of r-RM

In the following, we will provide and prove correct two utilization bounds for the r-RM scheduling algorithm, Theorem 4 and 5, respectively.

Theorem 4. A task system \( \tau \) is schedulable if:

\[
U_{\text{sum}}(\tau) \leq mU_B - (m - 1)U_{\text{max}}(\tau) \tag{1}
\]

Proof. In order to prove this theorem, we will first determine a necessary condition for such a system to be unschedulable. Its contraposition yields a sufficient condition for the given system to be schedulable.

Consider that a given job \((e, p)\) is not schedulable. This means that: \( \forall j : U_j > U_B - \frac{e}{p} \). Summing over all the processors in the platform, we have:

\[
\sum_{j=1}^{m} U_j > mU_B - m\frac{e}{p}.
\]

Given that \( \sum_{j=1}^{m} U_j \leq U_{\text{sum}}(\tau) - \frac{e}{p} \) (each task in the system may have at most one job active at each instant and the task whose job is being assigned does not contribute to the current utilization of any processor). Also taking into account that \( \frac{e}{p} \leq U_{\text{max}}(\tau) \), we can see that:

\[
U_{\text{sum}}(\tau) - \frac{e}{p} > mU_B - m\frac{e}{p} \iff
U_{\text{sum}}(\tau) > mU_B - (m - 1)\frac{e}{p} \Rightarrow
U_{\text{sum}}(\tau) > mU_B - (m - 1)U_{\text{max}}(\tau).
\]

We can conclude in the following that \( \tau \) is schedulable according to r-RM if \( U_{\text{sum}}(\tau) \leq mU_B - (m - 1)U_{\text{max}}(\tau) \).
This utilization bound is pessimistic in the case of systems containing heavy tasks. The following theorem provides a second utilization bound.

**Theorem 5.** A task system $\tau$ with $n_h$ heavy tasks and $n - n_h$ light tasks is schedulable by r-RM if:

$$U_{\text{sum}}(\tau) \leq (m - n_h)U_B - (m - n_h - 1)\frac{1}{2} + (n_h - 1)U_B + U_{\text{max}}(\tau)$$

where $U_B$ is a uniprocessor utilization bound (a constant $> 0.5$).

Proof. In the following, $\tau'$ represents the subset of light tasks of $\tau$. According to Theorem 4, $\tau'$ is schedulable if:

$$U_{\text{sum}}(\tau') \leq (m - n_h)U_B - (m - n_h - 1)U_{\text{max}}(\tau').$$

Taking into account that $U_{\text{max}}(\tau') \leq \frac{1}{2}$ we can conclude that the system is not schedulable if:

$$U_{\text{sum}}(\tau') > (m - n_h)U_B - (m - n_h - 1)\frac{1}{2} + U_{\text{max}}(\tau).$$

(2)

All the remaining tasks in $\tau$ are heavy ($u_i \geq 0.5$). In the best case, $(n_h - 1)$ of these tasks have a utilization of $\frac{1}{2}$ and only one has $U_{\text{max}}(\tau)$.

We have that

$$U_{\text{sum}}(\tau) \geq U_{\text{sum}}(\tau') + (n_h - 1)\frac{1}{2} + U_{\text{max}}(\tau).$$

So,

$$U_{\text{sum}}(\tau) > (m - n_h)U_B - (m - n_h - 1)\frac{1}{2} + U_{\text{max}}(\tau)$$

$$\Rightarrow U_{\text{sum}}(\tau) > (m - n_h)U_B - \frac{m}{2} + n_h + U_{\text{max}}(\tau).$$

(4)

Since $U_B > 0.5$

$$U_{\text{sum}}(\tau) > (m - n_h)\frac{1}{2} - \frac{m}{2} + n_h + U_{\text{max}}(\tau)$$

$$\Rightarrow U_{\text{sum}}(\tau) > \frac{n_h}{2} + U_{\text{max}}(\tau).$$

(5)

From Equations 4 and 6, we can conclude that for a system to be schedulable, it is sufficient to verify the following equation:

$$U_{\text{sum}}(\tau) \leq \alpha,$$

where

$$\alpha = \max\{(m - n_h)U_B - \frac{m}{2} + n_h + U_{\text{max}}(\tau); \frac{n_h}{2} + U_{\text{max}}(\tau)\}.\tag{7}$$

**Theorem 6.** If we have only light tasks (i.e. $n_h = 0$ \land $U_{\text{max}} < 0.5$) Theorem 4 dominates Theorem 5.

Proof. In this particular case ($n_h = 0$ \land $U_{\text{max}} < 0.5$)

$$\alpha = \max\{mU_B - \frac{m}{2} + U_{\text{max}}(\tau); U_{\text{max}}(\tau)\}$$

since $U_B > 0.5$ we have $mU_B - \frac{m}{2} > 0$ thus

$$\alpha = mU_B - \frac{m}{2} + U_{\text{max}}(\tau).$$

Now we will show that $\alpha = mU_B - \frac{m}{2} + U_{\text{max}}(\tau) < mU_B - (m - 1)U_{\text{max}}(\tau)$ (RHS of Equation 1).

Indeed since $U_{\text{max}}(\tau) < 0.5$ we have that $mU_B + U_{\text{max}}(\tau) - \frac{m}{2} < mU_B + U_{\text{max}}(\tau) - mU_{\text{max}}(\tau) = mU_B - (m - 1)U_{\text{max}}(\tau).$ \hfill ∎

From Theorems 4-6, we can conclude that a system $\tau$ is schedulable if:

$$U_{\text{sum}}(\tau) \leq mU_B - (m - 1)U_{\text{max}}(\tau), \text{ if } U_{\text{max}}(\tau) \leq 0.5,$$

$$U_{\text{sum}}(\tau) \leq \alpha, \text{ if } U_{\text{max}}(\tau) > 0.5$$

(8)

with $\alpha$ as defined by Equation 7.

### 6.3 Evaluation of r-RM and r-EDF

A classic way to analyze a schedulability test consists in determining the worst-case resource augmentation ratio that defines the limited amount of additional resources needed for a given sufficient schedulability test to match the performance of an exact feasibility test. The resource augmentation will be expressed as the worst-case speedup factor of each processor in the platform. A necessary condition for a real-time task set $\tau$ to be feasible upon $m$ unit capacity processors is $U_{\text{sum}}(\tau) \leq m$ and $U_{\text{max}}(\tau) \leq 1$. Theorem 4 states a sufficient schedulability condition for a task set to be schedulable by r-RM: $U_{\text{sum}}(\tau) \leq mU_B - (m - 1)U_{\text{max}}(\tau).$

We now present the resource augmentation approximation ratio for this sufficient schedulability test given by Theorem 4.

**Theorem 7.** Any real-time task set $\tau$ that is feasible upon $m$ unit capacity processors (with $m > 1$) is guaranteed to satisfy the condition of Theorem 4 on an $m$-processor platform where each processor has speed $\frac{2m - 1}{m}.\ln(2)$.

Proof. Let $\tau$ be a real-time task set that is feasible on a platform $\Pi$ with $m > 1$ unit capacity processors. Consider the platform $\Pi'$ based on platform $\Pi$ where each processor has been sped up by a factor $s \geq 1$. Let $\tau'$ be the task set $\tau$ with execution time normalized to the speed $s$: for every $\tau_i' \in \tau', \ p_i' = p_i$ and $e_i' = \frac{e_i}{s}$. We have:

$$U_{\text{sum}}(\tau') = s \times U_{\text{sum}}(\tau)$$

$$U_{\text{max}}(\tau') = s \times U_{\text{max}}(\tau)$$

Since $\tau$ is feasible then $U_{\text{sum}}(\tau) \leq m$ and $U_{\text{max}}(\tau) \leq 1$. Starting from the utilization bound given by Theorem 4, where $U_B$ is the maximum utilization bound of the fixed-task uniprocessor schedulability test:

$$U_{\text{sum}}(\tau) \leq mU_B - (m - 1)U_{\text{max}}(\tau')$$

$$s \times U_{\text{sum}}(\tau) \leq s \times (mU_B - (m - 1)U_{\text{max}}(\tau))$$

$$s \times U_{\text{sum}}(\tau) + (m - 1)U_{\text{max}}(\tau) \leq s \times mU_B$$

Thus, the smallest possible speedup factor corresponds to $U_{\text{sum}}(\tau) = m$ and $U_{\text{max}}(\tau) = 1$. Thus, we set the speedup factor as $s = \frac{mU_B}{m - 1}$. As a consequence, we set:

$$U_{\text{sum}}(\tau') = \frac{mU_B}{2m - 1} \times U_{\text{sum}}(\tau)$$

$$U_{\text{max}}(\tau') = \frac{mU_B}{2m - 1} \times U_{\text{max}}(\tau)$$

We now show that if the condition defined given by Theorem 4 on platform $\Pi'$ holds for the task set $\tau'$, then the task set $\tau$ is feasible.

$$U_{\text{sum}}(\tau') \leq mU_B - (m - 1)U_{\text{max}}(\tau')$$

$$U_{\text{sum}}(\tau') \leq mU_B - (m - 1)\frac{mU_B}{2m - 1} - U_{\text{max}}(\tau)$$

The maximum utilization bound for $\Pi'$ will be defined when the right-hand side of the previous inequality is minimized. It is achieved when $U_{\text{max}}(\tau) = 1$. Hence,
Thus, the task set $\tau$ is feasible upon the platform II. Using Liu and Layland’s maximum utilization bound defined as $U_B = n(2\ln n - 1) \geq \ln(2)$, then we achieve the worst-case speedup factor of $s = \frac{\ln(2)}{m\ln(n)}$. \hfill \Box

When $m$ approaches infinity the previous resource augmentation ratio approaches $\frac{2}{\ln(2)} \approx 2.88$. The factor of 2 is due to the multiprocessor platform with restricted-migration whereas the factor of $1/\ln(2)$ is inherited from the fixed-task priority schedulability analysis (i.e. Liu and Layland’s maximum utilization bound). Note that for determining the resource augmentation approximation ratio of $r$-rm, we used an exact feasibility algorithm (e.g. an algorithm of the PFAIR family). We leave as an open issue the tightness of the resource augmentation approximation ratio.

To the best of our knowledge, no resource augmentation ratio is known for $r$-EDF. We next show that the previous proof can be quite easily adapted to define the worst-case resource augmentation ratio of $r$-EDF.

**Theorem 8.** Any real-time task $\tau$ that is feasible upon $m$ unit capacity processors, $m > 1$, is guaranteed to satisfy the schedulability conditions of $r$-EDF stated (Theorem 3 in [8]) on an $m$-processor platform where each processor has speed $2 - \frac{1}{m}$.

**Proof.** If we set $U_B = 1$ then the condition of Theorem 4 leads to Theorem 4 of [8] (i.e. $U_{\text{sum}}(\tau) \leq m - (m - 1)U_{\text{max}}(\tau)$). The previous analysis presented in the proof of Theorem 4 can be also easily adapted by removing the maximum utilization bound $U_B = \ln(2)$ inherited from the fixed-task priority schedulability test and replacing it by the maximum utilization bound of $U_B = 1$ known for uniprocessor EDF feasibility in the whole proof. It is then easy to see that the worst-case resource augmentation ratio of $r$-EDF is $2 - \frac{1}{m}$. \hfill \Box

As for $r$-RM, we do not know if the previous resource augmentation approximation ratio of $r$-EDF is tight or not. The optimal speedup factor for global EDF is $2 - \frac{1}{n}$ (i.e. the bound is tight) [35], but we also know that global and partitioned scheduling algorithms are incomparable [12, 25].

### 7. CONCLUSION

In this research we considered the scheduling of real-time periodic tasks on multiprocessors with restricted-migration. We studied and shown the benefit of (offline) job partition, we showed that the (offline) job partitioning strictly dominates both (popular) partitioned and restricted-migration scheduling. We considered $r$-RM, a restricted-migration variant of RM, we proposed a sufficient schedulability test and we provided a speedup factor of $\frac{2m-1}{m(m-1)}$.

### 8. REFERENCES


