Reducing Preemptions and Migrations in Real-Time Multiprocessor Scheduling Algorithms by Releasing the Fairness

Geoffrey Nelissen[1], Vandy Berten, Joël Goossens, Dragomir Milojevic
Parallel Architectures for Real-Time Systems (PARTS) Research Center
Université Libre de Bruxelles (ULB)
Brussels, Belgium
{Geoffrey.Nelissen, Vandy.Berten, Joel.Goossens, Dragomir.Milojevic}@ulb.ac.be

***Abstract*—Over the past two decades, numerous optimal scheduling algorithms for real-time systems on multiprocessor platforms have been proposed for the Liu & Layland task model. However, recent studies showed that even if optimal algorithms can theoretically schedule any feasible task set, suboptimal algorithms usually perform better when executed on real computation platforms. This can be explained by the runtime overheads that such optimal algorithms induce.**

We have observed that all current optimal online multiprocessor real-time scheduling algorithms are (completely or partially) based on the notion of fairness. The respect of this fairness can be the cause of numerous preemptions and migrations.

We therefore propose a new algorithm —named U-EDF— which releases the property of fairness and instead use an EDF-like scheduling policy.

The simulation results are really encouraging since they show that, in average, U-EDF produces less than one preemption and one migration per job released during the schedule. Furthermore, we strongly believe in the optimality of our algorithm since all tested task sets were correctly scheduled under U-EDF.

I. Introduction

Over the past two decades, numerous optimal scheduling algorithms for real-time systems on multiprocessor platforms have been proposed for the Liu & Layland task model [1]. However, they have barely been used in industrial and commercial applications so far, even though their advantages are quite obvious: e.g., they maximize the utilization of the computation platform (i.e., the worst case achievable utilization bound is m, the number of processors). Consequently, more complex functionalities can be implemented on a given platform or, on the other hand, the number or the computing power of processors can be lower for a given application. The cost, the size and the power consumption of the platform could be therefore reduced.

We know that the adoption of new algorithms in the industry appears as a rather slow process if they are not hard proven in real conditions. We know also that most of the time there is a huge gap between the theory and practice/application —often due to concrete and practical factors.

[1]Supported by the Belgian National Science Foundation (F.N.R.S.) under a F.R.I.A. grant.

Recent studies showed that even if optimal algorithms can theoretically schedule any feasible task set, *suboptimal* algorithms usually perform better when executed on real computation platforms [2]. This can be explained by the runtime overheads (e.g., task preemption which induces cache memory refresh, job migration which induces context migration) that such optimal algorithms require. Hence, algorithms with numerous preemptions and migrations, while theoretically optimal or high-performing, are not viable in real applications.

On the other hand, suboptimal algorithms generally have low utilization bounds, e.g., the worst case achievable total utilization is only 65% for the semi-partitioned algorithm EDDP [3]. We therefore need more powerful platforms to implement applications using such suboptimal schedulers.

The challenge we address in this research is to design a scheduling algorithm with high utilization bound but with few preemptions and migrations.

To the best of our knowledge, all online optimal scheduling algorithms for multiprocessor platforms are (completely or partially) based on the notion of *fairness*. On the other hand, suboptimal algorithms that perform better than optimal ones are usually based on simple scheduling policies such as EDF [3], [4], [5].

This research. In this work, we will present a technique to significantly reduce the number of preemptions and migrations by sacrificing the fairness property required in optimal scheduler for the Liu and Layland task model upon identical multiprocessors. We propose an online scheduling algorithm —named U-EDF— which is not fair in its actual schedule (while using some fairness notions in its internal structure) and enables a significant reduction of the amount of preemptions and migrations in comparison to the best optimal scheduling algorithms.

II. Model and Related Work

In this paper, we study the problem of scheduling a set τ of n independent strictly periodic tasks with implicit deadlines on a platform of m identical processors. Each task τ_i is

1533-2306/11 $26.00 © 2011 IEEE
DOI 10.1109/RTCSA.2011.57

characterized by a worst case execution time C_i and a period T_i. That is, the arrivals of two successive jobs of τ_i are separated by T_i time units and each job must be executed for C_i time units before the next arrival. We define the utilization factor of τ_i as $U_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$. It measures the proportion of time that τ_i must execute on average to meet its deadline. The total utilization of the system is the sum of all the task utilizations divided by the number of processors i.e., $U_{tot} \stackrel{\text{def}}{=} \frac{\sum_{\tau_i \in \tau} U_i}{m}$.

A. Proportionate Fairness

The first optimal real-time scheduling algorithm for multiprocessor platforms was designed by Baruah *et al.* in 1993 [6]. The main idea of this algorithm was to follow as closely as possible the *fluid schedule*. That is, at any instant t, any task τ_i must have been executed $U_i \times t$ units of time from the start of the schedule. Notice that, in a fluid schedule, every task should be executed permanently at a rate equal to its utilization factor. It is therefore impossible to implement such a fluid schedule if the number of tasks is greater than the number of processors.

In Proportionate Fair (PFair) schedulers, the time is therefore discretized and the tasks can only be executed during an integer number of quanta. The time quanta are then *fairly* distributed between the tasks so that at any time t, the difference between the execution time of every task τ_i and the fluid schedule is smaller than 1 quantum of time.

This principle allowed to design numerous optimal multiprocessor real-time scheduling algorithms such as PF [6], PD [7], PD^2 [8] and ER-PD [9].

Unfortunately, PFair algorithms can potentially incur m preemptions and m migrations after each time quantum. Hence, they can potentially cause enormous runtime overheads which in turn reduce the schedulability of the task systems.

B. DP-Fairness and Boundary Fairness

Recently, many algorithms have made use of the Deadline Partitioning Fairness technique (DP-Fairness) [10], [11], [12], [13]. These algorithms ensure the fairness only at the deadlines of the tasks. The time is therefore divided in time slices bounded by two successive deadlines (not necessarily from the same task). At the k^{th} deadline instant d_k, any task τ_i must have been executed $U_i \times d_k$ time units from the start of the schedule. That is, every task must respect the fairness property at a task deadline. However, the schedule must not necessarily be fair between the two successive deadlines. To ensure this property, each task τ_i has to be executed an amount of time equals to $U_i \times (d_k - d_{k-1})$ within the time slice extending from the $(k-1)^{th}$ deadline instant d_{k-1} to the k^{th} deadline d_k.

Zhu *et al.* proposed a similar approach named Boundary Fairness (BFairness) and based on a discrete time model [14]. The main advantage of their algorithm on the DP-Fair model is that it does not allow a task to be scheduled during a time smaller than one quantum of time. That is, we can be sure that a minimum amount of work will be executed by the task before

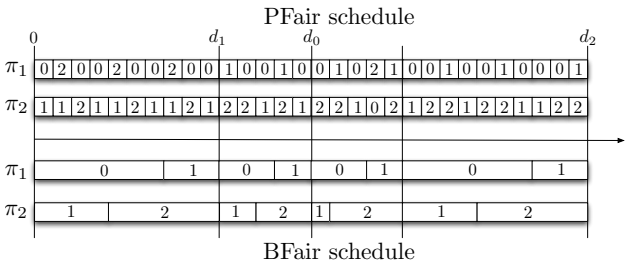

Fig. 1: PFair and BFair schedules on two processors of a system of tasks composed of three tasks τ_0, τ_1 and τ_2 with periods and worst case execution times defined as follows: $T_0 = 15$, $T_1 = 10$, $T_2 = 30$, $C_0 = 10$, $C_1 = 7$ and $C_2 = 19$.

to be preempted. Therefore, we can, for instance, choose a quantum of time to be significantly larger than a preemption and a migration cost.

By ensuring the fairness only at the deadline of tasks, best DP-Fair and BFair algorithms significantly reduce the number of preemptions and migrations in comparison to the PFair algorithms. Fig. 1 shows the correspondence between a PFair and a BFair schedule. By simply regrouping all the time units of a same task executed within a time slice, it is possible to highly reduce the number of preemptions and migrations of this task between the two boundaries. This property is illustrated in Fig. 1 where, for instance, the task τ_0 is subject to 4 preemptions in the BFair schedule instead of 11 in the PFair schedule within an hyper-period (i.e., the least common multiple of all the task periods).

C. The EKG Approach

We now introduce the optimal version of the EKG algorithm proposed by Andersson *et al.* in [15]. EKG makes use of a parameter k which allows to divide the task set into subsets with a total utilization smaller than k. Each subset is then scheduled on its own cluster of k processors. EKG is optimal when the parameter k is equal to the number of processors m. In the following, we will thereby suppose that $k = m$.

To explain the EKG mechanism, we will first introduce the simplest DP-Fair scheduler (i.e., when we consider a continuous time). This algorithm is named DP-Wrap [10]. Notice that, from a chronology perspective, DP-Wrap was proposed after EKG. However, it is easier to understand EKG by explaining DP-Wrap first.

DP-Wrap works in two phases. First, it assigns the tasks among the processors using a next fit heuristic. Tasks are assigned to a processor as long as the capacity on this processor is not exhausted. Let $u_{i,j}$ denote the *local utilization* of task τ_i assigned on processor π_j. Assuming that τ_i is the next task to assign, if there is no more available capacity on processors π_1 to π_{j-1}, then if the remaining capacity on π_j is greater than U_i, τ_i is statically assigned on π_j (i.e., $u_{i,j} = U_i$ and $u_{i,k} = 0 \ \forall k \neq j$). On the other hand, if the remaining capacity on π_j is smaller than U_i then τ_i becomes a migratory

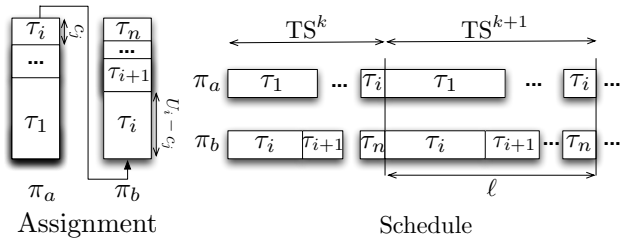

Fig. 2: Task assignment and schedule under DP-Wrap.

task and is split between the processors π_j and π_{j+1}. That is, if the remaining capacity on π_j is c_j then $u_{i,j} = c_j$ and $u_{i,j+1} = U_i - c_j$ (see Fig. 2).

The second phase consists in the schedule itself. Each task is executed within each time slice, during a time proportional to its local utilization on each processor. That is, if the time slice length is ℓ then τ_i is scheduled $u_{i,j} \times \ell$ time units on π_j. All tasks must therefore be scheduled in all time slices.

EKG outperforms this approach by regrouping all non-migratory tasks statically assigned to the same processor π_j in one *supertask*[2] S_j [15]. This supertask is scheduled in a DPFair manner with a utilization U_{S_j} equals to the sum of the utilizations of its component tasks. Then, whenever this supertask S_j is selected to be executed, EDF is used to decide which component task has to actually be scheduled on π_j.

EKG can therefore be seen as a hierarchical scheduler. On one hand, the supertasks are scheduled under a DP-Fair policy, and EDF is used to schedule the component tasks on the other hand.

In the worst case, the number of preemptions with EKG is $2 \times m$ preemptions per time slice rather than $n - 1$ preemptions per time slice with DP-Wrap [10], [15]. The number of migrations is identical in both algorithms and equal to m migrations per time slice in the worst case scenario.

III. U-EDF: Algorithm Description

A. Main Idea

We have seen in the previous section that, by first reducing the quality of the fairness during the schedule (e.g. DP-Fair algorithms only ensure the fairness at the task deadlines rather than after each time quantum) and then by introducing EDF in the scheduling policy (e.g. EKG), the number of preemptions and migrations can significantly be improved.

EKG is more efficient than DP-Wrap since it uses EDF for the schedule of the non-migratory tasks. However, both supertasks and migratory tasks still follow some fairness properties at the boundaries of the times slices.

To push ahead the reasoning of EKG, we propose to release any notion of fairness in the schedule of all the tasks and instead use an EDF-based scheduling policy to more drastically reduce preemptions and migrations. However, if we want to keep a high utilization bound, a careful assignment of the tasks among the processors must be realized repeatedly at some well chosen points during the schedule.

[2]The term *supertask* was first introduced by Moir *et al.* in [16] to denote a group of tasks scheduled as a unique task. Note that this appelation was not used in [15]. However, we will use this term since it is appropriate in the EKG context. In other works, groups of tasks have also been named *servers* (e.g. [17]).

	τ_1	τ_2	τ_3	τ_4
C_i	5	12	17	30
T_i	15	20	30	60

TABLE I: Task parameters for Example 1

To illustrate our idea, we will use an example showing how we can improve the schedule of the *first* job of every task.

***Example** 1:* We consider four tasks τ_1 to τ_4 and two processors π_1 and π_2. The characteristics of these tasks are given in Table I, and Fig. 3(a) depicts the schedule produced by DP-Wrap when the tasks have been assigned among the processors in an *increasing deadline order*. Now, let us consider this schedule as two independent parts. On one hand, we have the schedule of the first job of each task (white parts on Fig. 3(a)) and on the other hand we have the schedule of all the other jobs (striped parts on Fig. 3(a)).

We will now adapt the schedule of the white parts (first job of each task) in order to reduce the amount of preemptions, without changing the schedule of subsequent jobs (striped parts).

Fig. 3(b) shows the same schedule as Fig. 3(a) but separating the schedule of the white and striped parts.

Now, we will use EDF on each individual processor to schedule the first job of each task (white parts) in the portions of time that are not yet occupied by the other jobs (see Fig. 3(c)). Notice that the first time slice is entirely free and therefore only EDF is applied in this slice. Hence, the schedule is completely unfair within $[0, d_1)$ (τ_4 is not scheduled at all for instance). However, the system is still feasible after d_1 since, as we can see on Fig. 3(c), there is still a valid schedule after d_1.

Using this approach, we have reduced the number of preemptions in the first time slice where only the first jobs of the tasks are executed. To be able to continue to decrease the number of preemptions and migrations in future time slices, we will need to compute a new assignment of the tasks among the processors after d_1. This mechanism will be explained in the next section.

B. Formal Description

We first introduce some notations that we will use in the remaining of this paper. Then, we will formally present our algorithm named U-EDF (which stands for Unfair scheduling algorithm based on EDF).

Since we are working with implicit deadline tasks, each task has at most one active job at any time t. Therefore, we define the **current job of τ_i at time t** as the instance of τ_i, which

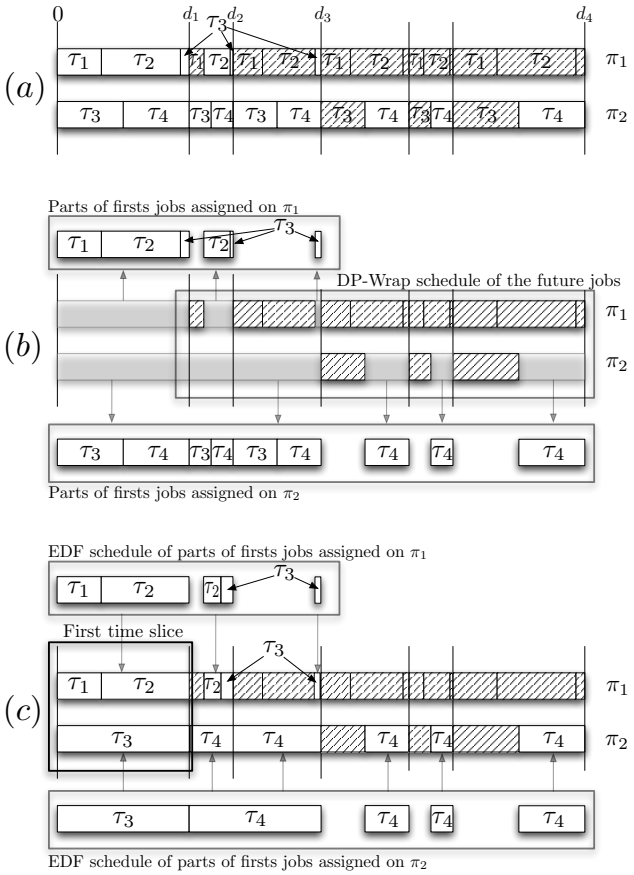

Fig. 3: Construction of a light-preemption scheduling algorithm with a high utilization bound.

arrived before or at time t, and having its deadline strictly after t.

From this definition, we can, without any ambiguity, denote by $d_i(t)$ the absolute deadline of the current job of τ_i at time t. We then have $d_i(t) > t$.

We also denote by $d^{\min}(t)$ the very first deadline after t, i.e.,

$$d^{\min}(t) \stackrel{\text{def}}{=} \min_i \{d_i(t)\}$$

At any instant t, we define the **(worst case) remaining execution time $r_i(t)$ of τ_i** as the amount of time units that the current job of τ_i at time t has still to execute before its next absolute deadline $d_i(t)$.

We finally define the **current job assignment**, denoted $q_{i,j}(t)$ as the part of $r_i(t)$ (expressed in time units) assigned on π_j during the interval $[t, d_i(t)]$. This concerns then only the current job of task τ_i at time t.

This assignment can vary during the schedule. That is, we can redefine the m values $q_{i,1}(t), \ldots, q_{i,m}(t)$ during a reassignment of the task among the processors.

Table II summarizes the notations used in this paper.

The U-EDF algorithm is based on three main principles:

C_i	Worst case execution time of task τ_i
T_i	Period of task τ_i
U_i	Utilization of τ_i: $U_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$
$d_i(t)$	Deadline of the current job of τ_i at time t
$d^{\min}(t)$	The very first deadline after t
$r_i(t)$	Worst case remaining execution time of τ_i at time t
$q_{i,j}(t)$	Part of $r_i(t)$ assigned on π_j during the interval $[t, d_i(t)]$

TABLE II: Summary of the notations used in this paper.

Principle 1: We divide the time in two different parts:

- $[t, d^{\min}(t)]$ i.e., the first time slice after t: the assignment is done in such a way that the system will be schedulable in $[t, d^{\min}(t)]$ using a variant of EDF avoiding parallelism (EDF-D, presented later in Section III-D).
- $[d^{\min}(t), \infty)$: the assignment is done in such a way to preserve the feasibility after $d^{\min}(t)$.

Principle 2: For each task τ_i, we reserve $r_i(t)$ units of time before $d_i(t)$.
That is, we reserve enough time for τ_i such that it can complete its execution before its deadline.

Principle 3: For each task τ_i, we reserve a time proportionate to U_i in each time slice after $d_i(t)$, i.e., we reserve $U_i \times \ell$ units of time in each time slice of length ℓ.
That is, after $d_i(t)$, we apply the same approach as DP-Fair algorithms which are known to be optimal. The main idea is to keep a feasible schedule for τ_i after $d_i(t)$. It is important to understand that the time reservation is a virtual concept: reserving x units of time on π_j for τ_i does not mean that we will actually run τ_i for x units of time on the processor π_j. Somehow, the scheduler could modify the assignment before having executed the x time units.

Based on those principles, we build our algorithm in two phases. The first phase is the assignment, i.e., the choice of the values for $q_{i,j}(t)$, for each task and each processor. The second phase is the actual schedule of jobs, respecting this assignment (until a new assignment computation).

The U-EDF algorithm will take an assignment decision on *every time slice boundary*, i.e., on each job arrival.

If, at each such job arrival, we manage (i) to provide to each task enough time to finish its current job by its deadline, (ii) to reserve in each time slice after its deadline a time proportionate to its utilization factor, and (iii) that we moreover can prove that we are able to do that without parallelism, then our scheduling algorithm will be correct (i.e., all deadlines are respected without intra-job parallelism).

C. First phase: Assignment

As explained earlier, the first phase of our algorithm consists in choosing the assignment of each current job, i.e., for each task, how much work has to be reserved on each processor. We say "reserved" and not "performed", because as we do a job assignment on every time slice boundary, there is no guaranty that we will actually perform $q_{i,j}(t)$ units of time of τ_i on π_j.

In all the remainder of this paper, we will assume that the tasks are always ordered according to their absolute deadlines. That is,

$$\forall j,k \ \mid \ j<k : d_j(t) \leq d_k(t)$$

If two tasks have the same deadline, the tie can be broken arbitrarily. Tasks are therefore re-indexed repeatedly on the fly according to their current deadlines.

The assignment algorithm of U-EDF is presented in Algorithm 1. The value $q_{i,j}^{\max}(t)$ denotes the maximum amount of execution time that can be allocated to τ_i on π_j without any risk of intra-job parallelism during the schedule. That is, for every task τ_i, we assign on processor π_j the minimum between $q_{i,j}^{\max}(t)$ and the amount of remaining execution time of τ_i that has not yet been assigned on another processor.

Algorithm 1: Assignment Algorithm.

Input:
$TaskList :=$ list of the n tasks sorted by increasing absolute deadlines;
$t :=$ current time;

```
forall τ_i ∈ TaskList do
    for j := 1 to m do
        q_{i,j}(t) := min{q^max_{i,j}(t), r_i(t) − Σ_{j'<j} q_{i,j'}(t)};
    end
end
```

In order to compute $q_{i,j}^{\max}(t)$, we first need to define some intermediate quantities.

Definition 1: The **future jobs reservation**, denoted $\boldsymbol{u_{i,j}(t)}$ is defined as:

$$u_{i,j}(t) \stackrel{\text{def}}{=} \left[U^i - (j-1)\right]_0^1 - \left[U^{i-1} - (j-1)\right]_0^1$$

where $U^i = \sum_{i' \leq i} U_{i'}$ and $[x]_a^b \stackrel{\text{def}}{=} \max\{a, \min\{b, x\}\}$.

In U-EDF, the future reservation corresponds to the proportion of time reserved for future instances of τ_i, in each time slice after $d_i(t)$ (i.e., the "striped parts" of each task in Fig. 3). Computing those values boils down to align blocs of size U_i, and cut them in boxes of size 1 (as in DP-Wrap algorithm [10]). Fig. 4 illustrates how to obtain this value. If we assume that tasks are ordered according to their deadlines, we consider the sequence $U_1, U_2, \ldots, U_n$, split in boxes of size 1. The boundaries of the bloc corresponding to τ_i are then U^{i-1} and U^i, and in order to know how much of τ_i goes on to processor π_j, we have to compute how much of the interval $[U^{i-1}, U^i]$ overlaps the interval $[j-1, j]$. Therefore, $\left[U^i - (j-1)\right]_0^1$ is the distance between the $j-1$ and the right side of the interval U^i, and $\left[U^{i-1} - (j-1)\right]_0^1$ is the distance between $j-1$ and the left side of the interval U^{i-1}.

Definition 2: The **earlier tasks reservation**, denoted $\boldsymbol{\rho_{i,j}(t)}$ is defined as:

$$\rho_{i,j}(t) \stackrel{\text{def}}{=} \sum_{i'<i} \Big(q_{i',j}(t) + u_{i',j}(t) \times (d_i(t) - d_{i'}(t)) \Big)$$

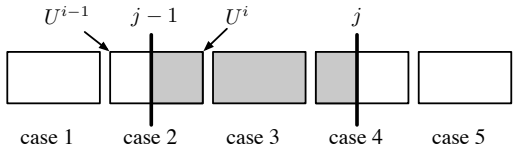

Fig. 4: Computation of $u_{i,j}(t)$.

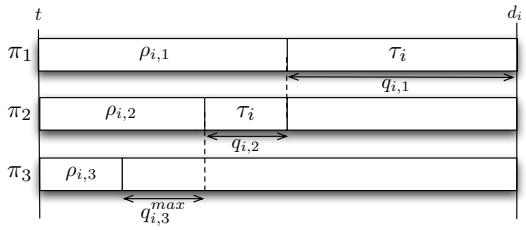

Fig. 5: Computation of $q_{i,j}^{\max}(t)$ for τ_i on processor π_3.

where $u_{i',j}(t)$ is the future job reservation as defined in Definition 1 and $q_{i',j}(t)$ is the assignment on π_j of tasks with lower deadlines than τ_i.

In our algorithm, this corresponds to the amount of work already reserved on processor π_j within $[t, d_i(t))$ for all the tasks with a deadline lower than $d_i(t)$. As we consider that tasks are ordered according to their deadlines, this corresponds to the tasks $\tau_{i'}$ with $i' < i$. For each task $\tau_{i'}$, we consider that $\tau_{i'}$ reserved $q_{i',j}(t)$ time units before $d_{i'}(t)$ (white part of each task in Fig. 3), plus an amount of time for its subsequent jobs arriving between $d_{i'}(t)$ and $d_i(t)$, i.e., $u_{i',j}(t) \times (d_{i'}(t) - d_i(t))$ (the striped parts in Fig. 3).

Definition 3: The **maximum reservation**, denoted $\boldsymbol{q_{i,j}^{\max}}$ is defined as:

$$q_{i,j}^{\max}(t) \stackrel{\text{def}}{=} (d_i(t) - t) - \rho_{i,j}(t) - \sum_{j'<j} q_{i,j'}(t)$$

where the value of $\rho_{i,j}(t)$ is given by Definition 2.

This can be seen as the maximum amount of work that τ_i could execute on π_j in the interval $[t, d_i(t))$ without parallelism. Indeed, we cannot allocate more than $(d_i(t) - t)$ time units on one processor within the interval $[t, d_i(t)]$. Therefore, if there already are $\rho_{i,j}$ time units reserved on π_j before the assignment of τ_i then we can give $(d_i(t) - t) - \rho_{i,j}(t)$ time units at most to τ_i (i.e., $q_{i,j}^{\max} \leq (d_i(t) - t) - \rho_{i,j}(t)$). Moreover, accordingly to Algorithm 1, τ_i could have already received some time $q_{i,j'}$ on processors with lower indexes (i.e., $j' < j$). The value of $q_{i,j}^{\max}(t)$ is minimal when these $q_{i,j'}$ time units are scheduled as in the example depicted on Fig. 5. Since intra-task parallelism is not allowed, the maximum amount of work that τ_i could execute on π_j under this situation is given by Definition 3.

With some simple arithmetics, it is possible to prove that $q_{i,j}^{\max}(t)$ can never be negative.

Using Definition 3 to compute the value of $q_{i,j}^{\max}$, we

can apply Algorithm 1 whenever we need to realize a new assignment of the tasks (i.e., at each job arrival).

D. Second phase: Schedule

Once the assignment of the tasks among the processors has been realized, we use an EDF-like algorithm named EDF-D to schedule the tasks on the processors within the first time slice. Then, a new assignment will be performed for the next time slice.

EDF-D (EDF with Delays) is a scheduling algorithm which behaves in the following way between two job assignments:

- We first denote by $\mathcal{E}_j(t)$ the set of eligible jobs on π_j at time t. A job is eligible on π_j at time t if (i) this job is not currently being run on a processor with a lower index, and (ii) there is still some work to perform for task τ_i on processor π_j according to the current assignment.
- At any time t, the scheduling algorithm schedules the *eligible* job with the earliest deadline.

In other words, EDF-D behaves the same way as EDF on each processor, except that as soon as a job is scheduled on some processor π_j, it is removed from the eligible jobs of any processor with a higher index.

Algorithm 2 summarizes the working process of U-EDF. Whenever, an event occurs in the system (i.e., a task arrival, a task deadline or an execution ending), Algorithm 2 is called. If the event corresponds to a task arrival, it means that we reached the end of the current time slice. Therefore, we recompute a new task assignment among the processors. Then, we use EDF-D to actually schedule the tasks on the platform.

Algorithm 2: U-EDF scheduling algorithm.

```
Data: t := current time
if t is the arrival time of a job then
    Recompute the assignment ;          // Algorithm 1
end
for j := 1 to m do
    // Use EDF-D
    Update E_j(t) ;
    Execute on π_j the task with the earliest deadline in E_j(t) ;
end
```

Notice that U-EDF behaves exactly as EDF when there is only one processor in the platform. Indeed, all tasks are obviously assigned to the same processor and therefore, a task cannot be delayed when using EDF-D since it never executes on another processor.

E. Correctness

In this section, we propose to prove that U-EDF produces a correct schedule (i.e., all deadlines are respected without intra-job parallelism) within each time slice if a valid U-EDF assignment holds at each job arrival (i.e., at each time slice boundary). Moreover, we strongly believe that U-EDF produces a valid assignment at each job arrival, which would imply that U-EDF is optimal for the schedule of periodic tasks with implicit deadlines. However, the proof of the optimality of U-EDF is beyond the scope of this paper which tries to show that significant improvements can be made on the amount of preemptions and migrations if we release the property of fairness and instead use an EDF-based scheduling policy. Furthermore, the simulation results presented in Section V will show that all tested task sets were schedulable under U-EDF.

Lemma 1: If EDF-D is applied during the time interval $(t_1, t_2]$ then there is no intra-job parallelism within this time interval.

Proof: This is a direct consequence of the definition of EDF-D. ■

Definition 4: Let EDF-D* be a variant of EDF-D where $\mathcal{E}_j(t)$ is re-defined as $\{1, \ldots, n\}$ (i.e., all jobs are always eligible). We define the **delay incurred by a task τ_i on π_j using EDF-D** as the time difference between the end time of the $q_{i,j}$ times units of τ_i on π_j using EDF-D, and the end time of the $q_{i,j}$ times units of τ_i on π_j using EDF-D*.

Property 1: If EDF-D is applied during the time interval $(t, d_i(t)]$ then the delay incurred by τ_i on π_j using EDF-D is at most equal to

$$\sum_{j'<j} q_{i,j'}(t)$$

Proof: A job of task τ_i can only be delayed by the same job running on processors with lower index. So, in the worst case, all the work for τ_i could be executed on processor with index lower than j before we run τ_i on π_j. ■

Definition 5 (Valid U-EDF Assignment): We will say that an U-EDF assignment is valid at time t, if, for every task τ_i in τ, we have successfully assigned the whole remaining execution time of τ_i on the platform processors. That is,

$$\forall \tau_i \in \tau : r_i(t) = \sum_{j=1}^{m} q_{i,j}(t)$$

Theorem 1: Let t_a be the very first arrival time of a job after t. If the U-EDF assignment is valid at time t and EDF-D is applied within $(t, t_a]$ then all jobs with a deadline such that $d_i(t) \leq t_a$ respect their deadlines without intra-job parallelism.

Proof: From Lemma 1, intra-job parallelism can never occurs in a time interval when EDF-D is applied to schedule the tasks within this time interval. Thereby, we have to prove that all jobs with a deadline at or before t_a respect their deadlines applying EDF-D. Hence, we must prove that the current job assignments $q_{i,j}(t)$ of these jobs are completely executed before d_i on all processors.

By contradiction, assume that the current job assignment of τ_i on processor π_j does not respect its deadline at time d_i. It means that

$$q_{i,j} + e_h + e_d > d_i - t \tag{1}$$

where e_h is the execution time of jobs with higher priorities than τ_i on processor π_j, and e_d is the delay incurred by τ_i on π_j using EDF-D (see Definition 4).

Since the tasks are ordered according to their current job deadlines, all current jobs of tasks $\{\tau_1, ..., \tau_i\}$ have thereby

their deadlines at or before d_i. According to EDF-D, the job with the smallest deadline is executed first (except if it is already running on a processor with a smaller index). Hence,

$$e_h \leq \sum_{i'<i} q_{i',j} \tag{2}$$

Notice that d_i is an instant at or before t_a which actually corresponds to the very first job arrival after t. Thereby, only the current active jobs of the tasks have to be taken into account in Equation 2. Successive jobs of tasks $\{\tau_1, ..., \tau_{i-1}\}$ cannot interact with the schedule of τ_i within $(t, t_a]$ since they are not yet released.

Furthermore, from Property 1, it yields

$$e_d \leq \sum_{j'<j} q_{i,j'} \tag{3}$$

Injecting Equations 2 and 3 in Equation 1, we get

$$q_{i,j} + \sum_{i'<i} q_{i',j} + \sum_{j'<j} q_{i,j'} > d_i - t$$

$$\Leftrightarrow \quad q_{i,j} > (d_i - t) - \sum_{i'<i} q_{i',j} - \sum_{j'<j} q_{i,j'}$$

From Definition 2, we have

$$\rho_{i,j} = \sum_{i'<i} \Big(q_{i',j} + u_{i',j} \times (d_i - d_{i'}) \Big)$$

leading to

$$\rho_{i,j} \geq \sum_{i'<i} q_{i',j}$$

Then,

$$q_{i,j} > (d_i - t) - \rho_{i,j} - \sum_{j'<j} q_{i,j'} = q_{i,j}^{\max}$$

which is a contradiction with Line 3 of the U-EDF assignment algorithm (Algorithm 1). ■

IV. Implementation Considerations

In this section, we propose two complementary improvements for the implementation of the basic U-EDF algorithm. While easy to implement, they allow to drastically improve the performance of U-EDF in terms of preemptions and migrations. However, they do not change in any point the working process of the algorithm.

A. Virtual Processing

If implemented exactly as described in Section III-B, U-EDF can cause unnecessary preemptions and migrations. Indeed, the remaining execution time of each task is assigned to multiple processors. Moreover, this assignment varies after each job arrival.

To mitigate this issue, we apply a similar approach to the *"switching technique"* proposed in [18]. That is, we can see the theoretical schedule produced by U-EDF as a *virtual schedule* executed on m *virtual processors*. These virtual processors do not represent anymore real physical processors. However, at any instant t, there is a bijection (i.e. a unique association in both directions) between the *virtual processors* (denoted π_j) and the *physical processors* (denoted $\mathcal{P}_k$).

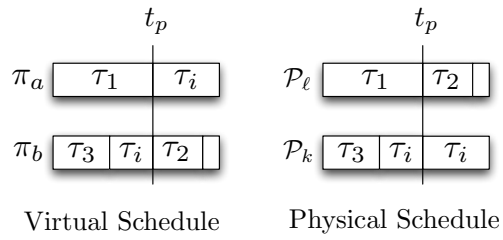

Fig. 6: Virtual and physical schedules when an instantaneous migration of task τ_i occurs at time t_p.

The main idea is that, if a schedule is correct at time t assuming a given virtual-physical processor association, then the schedule remains correct if we permute two physical processors. Indeed, the situation has not been changed for the scheduler since it schedules tasks on virtual processors and not on physical ones.

Using this technique, it is possible to modify the virtual-physical processor association such that if, at time t_p, a task τ_i instantly migrates from a processor π_b to π_a in the virtual schedule build by U-EDF then the physical processor $\mathcal{P}_k$ which was associated to π_b before t_p is associated to π_a after t_p (see Fig. 6). On the other hand, if π_a corresponded to $\mathcal{P}_\ell$ before t_p, then we will associate $\mathcal{P}_\ell$ to π_b after t_p. That is, we keep the task τ_i running on the same physical processor by permuting $\mathcal{P}_k$ and $\mathcal{P}_\ell$ in the virtual-physical processor association. We therefore save one unnecessary preemption in the real schedule actually executed on the physical platform.

To minimize the number of unnecessary preemptions, we need at most m permutations in the virtual-physical processor associations (one for each task executed on the platform).

Similarly, if a task τ_i is preempted when running on the physical processor $\mathcal{P}_k$, then when τ_i will be re-executed in the virtual schedule, we will try to associate its virtual processor with $\mathcal{P}_k$. Therefore, if the association is possible (i.e., the physical processor was not yet associated to another task), we save a migration in the real schedule executed on the physical platform.

We have applied this simple mechanism in all the simulations presented in Section V.

B. Clustering

As shortly explained in Section II-C, the original version of EKG proposes to partition the task set in clusters through the definition of a parameter k. This parameter defines the number of processors in each cluster. Then, a bin-packing algorithm can be applied to dispatch the tasks among the clusters such that the total utilization in each cluster does not exceed k. The EKG algorithm is then independently performed

on each cluster. Furthermore, in order to minimize the amount of preemptions and migrations, every task with a utilization greater than or equal to $\frac{k}{k+1}$ receives its own processor.

Using this approach, EKG can correctly schedule any task set with a utilization bound of $\left(\frac{k}{k+1} \cdot m\right)$ [15].

We used the same technique with U-EDF. That is, we always computed the smallest value for k such that a given task set was schedulable applying the previous condition. Then, we dispatched the tasks among the clusters using a *worst fit* algorithm.

This mechanism has been applied to both EKG and U-EDF in all our simulations. Notice that in the original version of EKG, a first fit algorithm was used. We preferred the worst fit one because of its ability to better distribute the workload among the clusters.

A more accurate choice of the number of processors in each cluster could be performed using the Qi *et al.* result in [19]. However, for its facility of comparison, we preferred the solution proposed in the EKG work. Indeed, for a given total utilization, all the task sets will have the same parameter k.

Divide the task system into clusters allows to reduce the number of reassignment points during the schedule (i.e., there are less job's arrival in each cluster). Therefore, the amount of migrations and preemptions caused by this reassignment process is reduced. Moreover, since there are less tasks interacting on each cluster, the number of preemptions is obviously improved.

V. Simulation Results

During our simulations, we compared the results obtained with U-EDF with the number of preemptions and migrations caused by DP-Wrap and EKG. Both are optimal online multiprocessor scheduling algorithms for periodic tasks with implicit deadlines. Under our knowledge, EKG is currently the most effective optimal multiprocessor online algorithm in terms of preemptions and migrations.

For all the simulations, we have applied the two implementations improvements presented in Section IV. That is, (i) for both EKG and U-EDF, we always partitioned the platform in clusters constituted of the smallest possible number of processors (see parameter k) whenever it was feasible (i.e., whenever the total utilization was smaller than 100%) and (ii) we always applied the virtual processing technique to reduce the amount of unnecessary preemptions and migrations.

For each experiment, we simulated 200 task sets on the entire hyper-period. For each task set, we have produced tasks with a utilization randomly chosen under a uniform distribution until the desired total utilization was obtained. We subsequently randomly choose the hyper-period under a uniform distribution on an interval extending from 100 to 1,000,000. Then, we decomposed the hyper-period in a product of prime numbers. Finally, for each task, we have randomly chosen its period such that it was a product of the prime numbers previously obtained. Using this technique, we can be sure that the hyper-period of the task system cannot be larger than 1,000,000 time units.

U-EDF succeeded in respecting the deadlines during the schedule of all the task sets simulated for our experiments. This result reinforces our belief in the U-EDF optimality.

Fig. 7 shows the average number of preemptions and migrations per job generated in the hyper-period when the number of processors in the platform varies from 2 to 24. We assumed a total utilization of 100% and a task utilization uniformly distributed within $[0.01, 0.99]$ for subfigures (a) and (b), and within $[0.01, 0.49]$ for subfigures (c) and (d).

We can observe that U-EDF performs drastically better than DP-Wrap and EKG for both the preemptions and migrations. In average, the gain can be as high as a factor 9 for the preemptions and a factor 3.5 for the migrations amount. In both cases, we get in average, less than 1 preemption and 1 migration per job during the whole schedule.

Fig. 8 presents our simulation results when the total utilization varies between 50% and 100% on a platform of 16 processors. The utilization of each task was randomly generated under a uniform distribution on the interval $[0.01, 0.49]$. Since the utilization of the platform was smaller than 100% during this experiment, the clustering technique presented in section IV was applied for both EKG and U-EDF. We observe that DP-Wrap has a relatively constant average number of preemptions and migrations per job. On the other hand, EKG sees his preemptions and migrations growing with the total utilization (i.e., with the number of processors in each cluster).

For U-EDF, the amount of preemptions decreases with the total utilization, while the amount of migrations increases. However, both stay smaller than 1 preemption/migration per job in average. Notice that the amount of migrations is close to 0 when the total utilization is smaller than 70% with these simulation parameters.

Moreover, for both DP-Wrap and EKG, the migrations and preemptions are strongly related to the number of time slices per job. Therefore, the variance on the average preemptions and migrations numbers is of the same order as the average value. On the other hand, U-EDF has a variance around 10% of the average value.

VI. Conclusion and Future Work

In this paper, we presented U-EDF, an online multiprocessor scheduling algorithm which does not require fairness and uses an EDF-based scheduling policy instead. We formally proved the correctness of our algorithm, and strongly believe in its optimality. The results of our simulations confirm this claim. By the way of these simulations, we showed one of the major interest of our algorithm: it strongly reduces the number of preemptions and migrations, compared to the state-of-the-art optimal online multiprocessor algorithms.

Indeed, as shown in our simulation results, for all our tested cases, we always have, in average, less than one preemption and one migration per job released during the hyper-period. In the best situations, the average preemptions per job are even

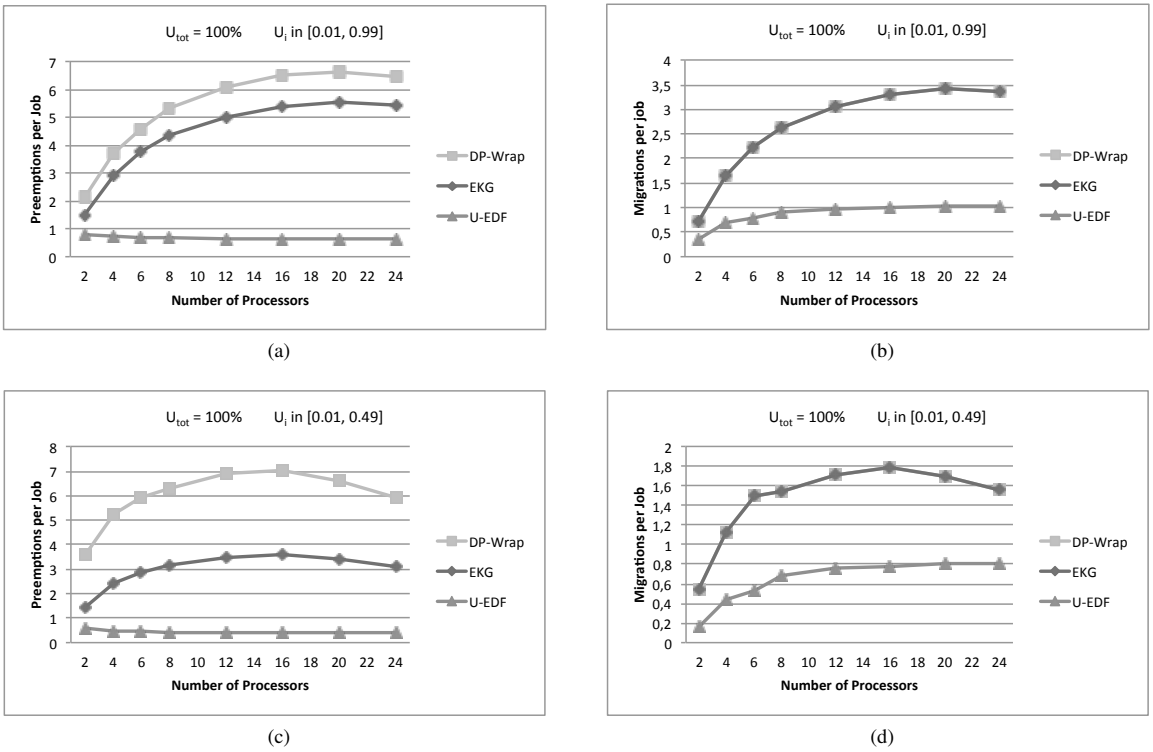

Fig. 7: Average number of preemptions [(a) and (c)] and migrations [(b) and (d)] per job when the total utilization equals 100% and the task utilization is uniformly distributed within 0.01 and 0.99 [(a) and (b)] or 0.01 and 0.49 [(c) and (d)].

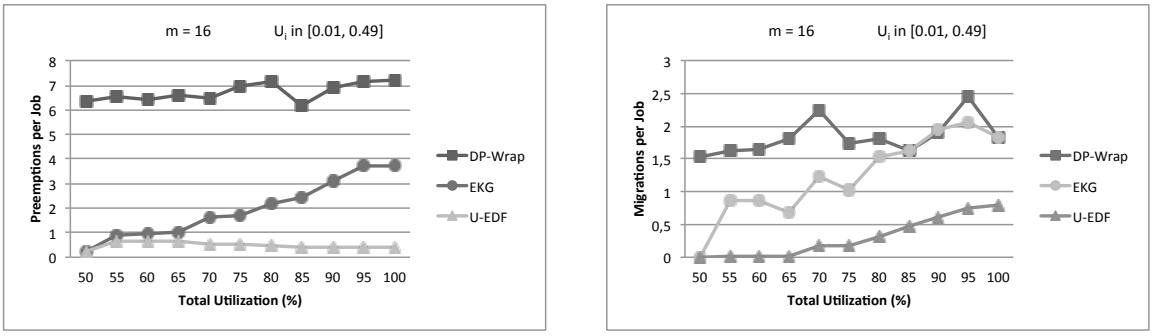

Fig. 8: Average amount of preemptions (a) and migrations (b) per job on 16 processors when the total utilization varies between 50 and 100% and the task utilization is uniformly distributed within 0.01 and 0.5.

less than 0.5 and the number of migrations could decrease under 0.2. In comparison, EKG, the best online multiprocessor scheduling algorithm known so far, incurs in average, between 1.5 and 5.5 preemptions and between 0.6 and 3.5 preemptions per job.

We consider several possible improvements of our work.

- We first plan to prove the optimality of U-EDF.
- Then, we would like to extend our algorithm to the schedule of sporadic tasks.
- We believe that some re-assignment points could be avoided which should reduce, once more, the number of preemptions and migrations.
- We would also like to study the practical issues that an implementation of our algorithm would raise.

REFERENCES

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

[2] A. Bastoni, B. Brandenburg, and J. Anderson, "Is Semi-Partitioned Scheduling Practical?" in *ECRTS'11*, Porto, Portugal, July 2011, to appear.

[3] S. Kato and N. Yamasaki, "Portioned EDF-based scheduling on multiprocessors," in *EMSOFT '08*. New York, USA: ACM, 2008, pp. 139–148.

[4] S. Kato, N. Yamasaki, and Y. Ishikawa, "Semi-partitioned scheduling of sporadic task systems on multiprocessors," in *ECRTS' 09*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 249–258.

[5] A. Burns, R. Davis, P. Wang, and F. Zhang, "Partitioned EDF scheduling for multiprocessors using a C=D scheme," in *RTNS '10*, 2010, pp. 169–178.

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: a notion of fairness in resource allocation," in *STOC '93*. ACM, 1993, pp. 345–354.

[7] S. K. Baruah, J. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *IPPS '95*. Washington, DC, USA: IEEE Computer Society, 1995, pp. 280–288.

[8] J. H. Anderson and A. Srinivasan, "Pfair scheduling: beyond periodic task systems," in *RTCSA '00*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 297–306.

[9] J. H. Anderson and A. Srinivasan, "Early-release fair scheduling," in *ECRTS '00*. Los Alamitos, CA, USA: IEEE Computer Society, 2000, pp. 35–43.

[10] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-Fair: A simple model for understanding optimal multiprocessor scheduling," in *ECRTS' 10*. IEEE Computer Society, July 2010, pp. 3–13.

[11] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *RTSS '06*. IEEE Computer Society, 2006, pp. 101–110.

[12] S. Funk and V. Nadadur, "LRE-TL: An optimal multiprocessor algorithm for sporadic task sets," in *RTNS' 09*, Paris, France, October 2009, pp. 159–168.

[13] K. Funaoka, S. Kato, and N. Yamasaki, "Work-conserving optimal real-time scheduling on multiprocessors," in *ECRTS'08*, 2008, pp. 13–22.

[14] D. Zhu, D. Mossé, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?" in *RTSS '03*. IEEE Computer Society, 2003, pp. 142–151.

[15] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," *RTCSA '06*, pp. 322–334, 2006.

[16] M. Moir and S. Ramamurthy, "Pfair scheduling of fixed and migrating periodic tasks on multiple resources," in *Proceedings of the 20th IEEE Real-Time Systems Symposium*. IEEE Computer Society, 1999, pp. 294–303.

[17] K. Bletsas and B. Andersson, "Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound," in *Proceedings of the 30th IEEE Real-Time Systems Symposium*. IEEE Computer Society, 2009, pp. 447–456.

[18] T. Megel, R. Sirdey, and V. David, "Minimizing task preemptions and migrations in multiprocessor optimal real-time schedules," in *RTSS'10*, December 2010, pp. 37–46.

[19] X. Qi, D. Zhu, and H. Aydin, "Cluster scheduling for real-time systems: utilization bounds and run-time overhead," *Real-Time Systems*, vol. 47, no. 3, pp. 253–284, 2011.