U-EDF: An Unfair but Optimal Multiprocessor Scheduling Algorithm for Sporadic Tasks

Geoffrey Nelissen[†], Vandy Berten[†], Vincent Nélis[§], Joël Goossens[†], Dragomir Milojevic[†]

[†] PARTS Research Centre
Université Libre de Bruxelles (ULB)
Brussels, Belgium

[§] CISTER-ISEP Research Centre
Polytechnic Institute of Porto (IPP)
Porto, Portugal

***Abstract*—A multiprocessor scheduling algorithm named U-EDF, was presented in [1] for the scheduling of periodic tasks with implicit deadlines. It was claimed that U-EDF is optimal for periodic tasks (i.e., it can meet all deadlines of every schedulable task set) and extensive simulations showed a drastic improvement in the number of task preemptions and migrations in comparison to state-of-the-art optimal algorithms. However, there was no proof of its optimality and U-EDF was not designed to schedule sporadic tasks.**

In this work, we propose a generalization of U-EDF for the scheduling of *sporadic* tasks with implicit deadlines, and we prove its *optimality*. Contrarily to all other existing optimal multiprocessor scheduling algorithms for sporadic tasks, U-EDF is not based on the fairness property. Instead, it extends the main principles of EDF so that it achieves optimality while benefiting from a substantial reduction in the number of preemptions and migrations.

I. Introduction

Nowadays, the current trend in the embedded industry is toward the utilization of multicore/multiprocessor platforms. Although such multicore platforms offer enhanced computational capabilities compared to traditional single core systems, the presence of several shared cores required a shift in the earlier design of scheduling algorithms. This introduced a plethora of new (and sometimes unexpected) challenges.

The question of "optimality" (i.e., ability of a particular scheduling algorithm to meet all deadlines of all tasks for any schedulable task set) quickly became one of the primary concerns in the design of multiprocessor scheduling algorithms.

Although schedulers like the Earliest Deadline First (EDF) algorithm are optimal on a single core platform, they were never extended to the scheduling of sporadic tasks on multiprocessor while keeping their optimality. However, despite this loss of optimality on multiprocessor, the simplicity of its scheduling rules on the one hand and the low scheduling overheads that it features on the other hand have motivated the research community to consider EDF as a reference. Several extensions of EDF to multiprocessor can be found in the literature and we refer the interested reader to [2]–[5] for further details.

Concurrently, during the past few years, we have witnessed the emergence of complex algorithms that fairly distribute the computing capacity of the platform to tasks [6]–[11]. Most of them, although shown to be optimal on multiprocessor, incur nevertheless high scheduling overheads, thereby limiting their possible deployment in real-world applications.

In this work, we propose an optimal extension of EDF to multiprocessor (named U-EDF) which is optimal for the scheduling of sporadic tasks with implicit deadlines. In contrast with other existing algorithms, U-EDF is not based on the fairness property, but instead uses the main principles of EDF so that it achieves optimality while benefiting from a substantial reduction in the number of preemptions and migrations compared to other optimal algorithms.

Related Works

The first optimal scheduling algorithm on multiprocessor platforms considered periodic task sets. This solution named Proportionate Fairness (PFair) was, as its name implies, based on a fair distribution of the processing capacity between tasks (i.e., each task being executed proportionally to its utilization factor) [12], [13]. Many PFair algorithms were designed over the years (e.g., PD [14], PD^2 [6], ER-PD [7]). However, although the fairness property ensures the optimality, it generates an extensive amount of preemptions and migrations, thereby limiting its applicability.

More recently, it has been noted that imposing a fair progress in task executions at each and every time instant t was too restrictive; imposing the fairness constraint only at task deadlines suffices to reach the optimality [8], [10]. From this observation, a new family of schedulers called DP-Fair [8] (or Boundary fair [10]) has been proposed. One may cite DP-Wrap [8], LLREF [9] or BF [10] as some algorithms following this refined concept.

Andersson, *et al.* went even further with the EKG algorithm, grouping tasks into servers [11]. Servers are scheduled in a DP-Fair manner but component tasks of each server are executed according to the EDF algorithm. It has been proven that EKG is optimal for the scheduling of periodic tasks.

Very recently, Regnier *et al.* presented the RUN algorithm [15], which reduces the multiprocessor scheduling problem to a uniprocessor schedule using a dualization technique. This algorithm is, to the best of our knowledge, the only solution different from ours (i.e., U-EDF), which does not use any fairness assumption but is nevertheless optimal for the scheduling of periodic tasks on multiprocessor. However, this approach does not handle sporadic tasks yet.

Finally, a previous version of the U-EDF algorithm was presented in [1] showing a drastic improvement of the number of preemptions and migrations due to the fairness release.

An interesting observation issued from the study of all the aforementioned algorithms, is the fact that the number of preemptions and migrations decreases as the fairness constraint is relaxed. The impact is even amplified when EDF rules are incorporated in the scheduler. That is, PFair algorithms cause much more overheads than Boundary Fair (or DP-Fair) approaches which in turn are more costly in terms of preemptions than EKG. Finally, it was shown that RUN and U-EDF dominate all these algorithms [1], [15].

Amongst the previously cited algorithms, some of them were successfully extended to the scheduling of sporadic tasks while keeping their optimality (e.g., PD^2 and ER-PD [6], DP-Wrap [8] and LLRE-TL [16]). On the other hand, RUN applies only to periodic task sets, and the sporadic generalizations of EKG [17], [18] are optimal only in some particular configurations which often cause an extensive amount of preemptions and migrations [19].

Hence, we can observe that, so far, the only scheduling algorithms that are optimal for the scheduling of periodic and *sporadic* tasks are based on the notion of fairness.

In this work, we extend the U-EDF algorithm to the scheduling of sporadic tasks with implicit deadlines and prove its optimality. Hence, we propose an optimal algorithm for sporadic tasks extending EDF to multiprocessor, in the sense that it reduces to EDF on uniprocessor platforms. As a consequence, U-EDF competes very well with other generalizations of EDF with respect to the number of preemptions and migrations, while preserving the optimality for the scheduling of periodic and sporadic tasks.

II. Model

We tackle the problem of scheduling a real-time system τ composed of n tasks $\{\tau_i, \tau_2, ..., \tau_n\}$ on m identical processors $\pi_1, \pi_2, \ldots, \pi_m$ assuming a *continuous-time* model. Each task $\tau_i \stackrel{\text{def}}{=} \langle C_i, T_i \rangle$ is a *sporadic* task characterized by a worst-case execution time (WCET) C_i and a minimum inter-arrival time T_i, with the interpretation that each task τ_i generates a potentially infinite sequence of jobs, each job has a WCET of C_i time units and every two consecutive jobs of τ_i are released at least T_i time units apart. A job of any task τ_i has to complete its execution before its deadline occurring T_i time units after its release, i.e., prior to the potential release of its next job. This requirement is known as the "implicit deadline" task model.

The utilization of a task τ_i is defined as $U_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$. We assume that U_i is not larger than 1 for every task τ_i in τ. Informally, the utilization of a task represents the highest percentage of time the task may use a processor (by releasing one job every T_i time units and executing each such job for C_i time units). The total utilization U of the system is defined as the sum of all the task utilizations: $U \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_i$. Roughly speaking, this quantity represents a lower-bound on the number of processors needed to successfully schedule the sporadic system τ.

We define the **active job** of τ_i at time t (if any) as *the* job of τ_i that arrived before or at time t and has its deadline strictly after t. From this definition we say that a task is active at time t if it has an active job at time t. Since we are considering implicit deadline tasks, each task has at most one active job at any time t. Hence, without any ambiguity we can refer to **the deadline $\boldsymbol{d_i(t)}$** of a task τ_i at time t as:

- the deadline of the current active job of τ_i at time t if τ_i is active;
- $d_i(t) = t$ if τ_i is not active at time t.

We say that a task τ_k has a higher priority than a task τ_i if and only if $d_k(t) < d_i(t)$ or $d_k(t) = d_i(t) \ \wedge \ k < i$. That is, tasks are prioritized according to EDF. Notice that following the definition of the task deadline at time t, tasks that are not active at time t have the highest priority. This particularity will be further motivated in the next section.

We define the **higher priority tasks set $\boldsymbol{\mathrm{hp}_i(t)}$** (respectively the lower priority tasks set $\mathrm{lp}_i(t)$) as the set of tasks with higher (respectively lower) priorities than task τ_i at time t (we recall that, in this particular case, terms job and task can be used interchangeably since each task has at most one active job at each instant t). Notice that, due to the definition of the deadline of a task at time t, the higher priority tasks set $\mathrm{hp}_i(t)$ of an active task always contains the tasks that are not active at time t (i.e., tasks with a deadline $d_x(t) = t$). Since EDF is used to prioritize tasks, the content of $\mathrm{hp}_i(t)$ and $\mathrm{lp}_i(t)$ may be altered by every new job release.

Finally, at any instant t, we define the (worst-case) **remaining execution time $\boldsymbol{\mathrm{ret}_i(t)}$** of τ_i as the (maximum) number of time units the active job of τ_i still has to execute before its deadline $d_i(t)$. If τ_i is not active at time t then $\mathrm{ret}_i(t) = 0$.

III. U-EDF: Algorithm Intuition

In order to give the intuition lying behind the U-EDF algorithm, in the remainder of this section, we distinguish between the scheduling of jobs and tasks. Scheduling a set of jobs assumes that all job release times are known at design time for the whole system lifespan. In contrast, scheduling sporadic tasks only assumes that the inter-arrival time between two job releases of a same task is known beforehand. That implies that the schedule cannot be constructed offline, since scheduling decisions depend on the actual job arrivals.

A. Scheduling jobs

The Earliest Deadline First (EDF) algorithm optimally schedules any feasible set of jobs on a uniprocessor platform [20]. As its name implies, this algorithm gives the highest priority to the active job with the earliest deadline.

In the next two paragraphs, we present two possible generalizations of EDF to multiprocessor (i.e., both reduce to EDF when the platform is composed of only one processor): the first generalization is said to be "vertical" and the second "horizontal".

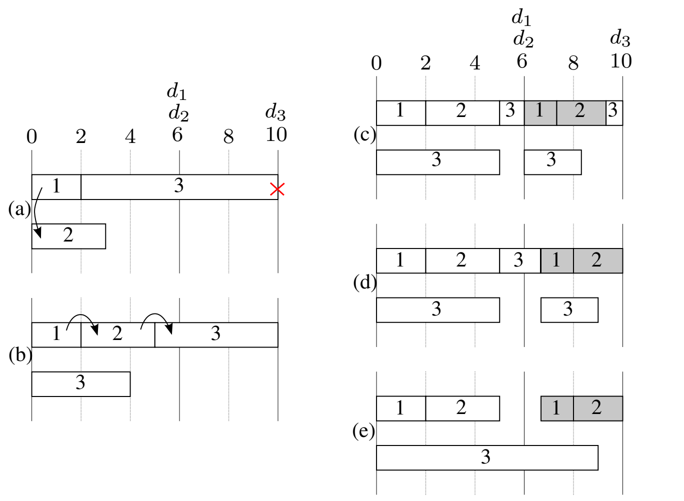


Fig. 1: (a-b) Two possible generalizations of EDF on multiprocessor: (a) Vertical generalization (G-EDF); (b) Horizontal generalization.
(c-d) Generation of the U-EDF schedule at time 0: (c) Allocation and reservation scheme at time 0. (d) Actual schedule if no new job arrives before 10. (e) Effective schedule with processor virtualisation.

1) Vertical Generalization: The straightforward generalization of EDF to multiprocessor platforms is certainly *global EDF* (G-EDF). With G-EDF, jobs are still prioritized according to their deadlines and, at any instant t, the m processors of the platform are allocated to the m active jobs of highest priorities. Informally, we qualify this allocation rule as being vertical for the following reason: at time 0, this G-EDF scheduling rule schedules the highest priority job on the first processor and likewise, the k^{th} highest priority job is scheduled on the k^{th} processor ($1 \leq k \leq m$). That is, in the usual representation of a schedule (see Figure 1), the m highest priority jobs are assigned to processors "from top to bottom".

Unfortunately, as shown in the following example, G-EDF is not optimal on multiprocessor.

Example: Let us consider a platform composed of two processors and the three jobs $J_1 = \langle 2, 6 \rangle, J_2 = \langle 3, 6 \rangle, J_3 = \langle 9, 10 \rangle$, where each tuple $\langle c, d \rangle$ describes the computation time c and the deadline d of a job (as we consider jobs and not tasks, we do not specify any period). We assume that all these jobs are released at time 0. Using G-EDF to schedule them, we can see on Figure 1(a) that J_3 misses its deadline. The reason being that G-EDF always executes the highest priority jobs as soon as possible without caring for what could happen afterward. Consequently, after executing J_1 and J_2, it is already hopeless for J_3 to meet its deadline unless J_3 could execute in parallel on both processors.

2) Horizontal Generalization: We should note that, EDF schedules highest priority jobs sequentially whereas G-EDF executes these same jobs in parallel. The rationale behind this parallel execution scheme is the will to execute highest priority jobs *as soon as* possible, in spite of the possibly negative impact on the future execution context.

But why should we hurry and start using more than one processor when it is not needed? An alternative approach consists in scheduling highest priority jobs (i.e., jobs with earliest deadlines) sequentially on the first processor, and start to allocate π_2 only if processor π_1 cannot afford to schedule all jobs while respecting their deadlines. The idea is to maximize the workload executed by the first processor before starting to use the second one. Similarly, we will maximize the workload assigned on both π_1 and π_2 before starting to allocate the third processor π_3.

Example: Consider the same platform and three jobs as in the previous example. The "horizontal" extension is illustrated on Figure 1(b). Similarly to Figure 1(a), the first job J_1 is executed on π_1 between instants 0 and 2. However, contrarily to G-EDF, the job J_2 is also allocated to processor π_1 between instants 2 and 5 (instead of processor π_2 in G-EDF). The objective of this new approach is to maximize the workload on the already-allocated processors (here π_1) before using another processor (here π_2). Five work units are then allotted to J_3 on π_1, up to its deadline d_3. Obviously, we cannot further schedule J_3 on π_1 without missing its deadline. Hence, we consider that π_1 is "full", and continue the execution of J_3 on π_2, from time 0 to 4.

In contrast to G-EDF which vertically extends EDF, this generalization of EDF may be seen as being *horizontal*. Indeed, as shown in Figure 1(b) and previously detailed in our example, after scheduling J_1 on π_1, we schedule J_2 on the same processor, directly after the completion of J_1. The second processor π_2 starts being allocated to a job only if this job cannot be entirely executed on π_1 by its deadline. Jobs are thus disposed one by one on the processors (following EDF), in a *horizontal* manner (similar to the algorithm proposed in [21]).

This horizontal approach is in line with the volition of EDF as it tries to maximize the amount of work executed by jobs of highest priorities before their respective deadlines. The difference between G-EDF and this "horizontal" generalization is that G-EDF is driven by the aforementioned will to execute jobs *as soon as* possible whereas the "horizontal" scheduler executes *as much as* possible on as few processors as possible.

B. Scheduling tasks

Unfortunately, the horizontal schedule presented above must be constructed offline. Furthermore, it needs to know every job arrival beforehand. However, exact release times cannot be known *a priori* when considering sporadic tasks. Therefore, we propose a new scheduling algorithm divided in two phases:

- First, whenever a new job is released, the "horizontal" technique is used to pre-assign the active jobs to processors. During this pre-assignation, extra time proportionate to the utilization factor U_i of every task τ_i is preventively reserved on the platform in any time interval following the deadline of every task τ_i. The idea is to save enough computing resources for the future jobs that might potentially be released and interfere with the schedule of the currently active jobs.

- Then, according to the allocation decided during the first phase, active jobs are executed on each processor, using a slight variation of EDF that avoids parallel execution of a same job (see Section IV-B).

Example: In Figure 1(c), once τ_1 has been assigned to π_1, we reserve (in gray on the figure) a fraction $1/3$ of the processor time ($U_1 = 1/3$) for the execution of future jobs of τ_1 that might be released after its current deadline. Then, τ_2 is assigned to π_1, and we reserve an half ($U_2 = 1/2$) of a processor for the possible future job it might release after its deadline at time 6. Finally, τ_3 is allocated by filling the gaps, first on processor π_1 up to its deadline, and then on π_2.

Note that during the scheduling phase, the time reserved for future jobs (in gray on Figure 1) will actually be utilized only if new jobs arrive. On the other hand, if no new job is released by τ_1 or τ_2, task τ_3 will just keep on running as shown in Figure 1(d).

This new scheduling algorithm composed of these two phases is called U-EDF.

It is important to understand that the schedule presented on Figure 1(d) might be modified if new jobs are released. Indeed, U-EDF recomputes the task allocation whenever a new job arrives in the system.

Furthermore, the reader probably noticed that, on Figure 1(d), τ_3 instantly migrates from π_2 to π_1 at time 5, and inversely at time 7. A virtual processor mechanism presented in [1] avoids these pointless migrations and produces the schedule illustrated in Figure 1(e).

At this point of the paper, we can understand why the deadline of a non-active task at time t is set to t. In this case every non-active task has the highest priority (i.e., tasks are prioritized using EDF and no task can have a current deadline smaller than t following the definition given in Section II). Since U-EDF allocates processor time to tasks in a decreasing priority order, it means that the first action taken by U-EDF is to reserve some time (proportionally to the utilization of non active tasks) for the scheduling of future jobs that non active tasks could possibly release after t. Therefore, we are certain to save enough time on processors for the schedule of future jobs of non active tasks before starting to allocate time to the active tasks.

IV. U-EDF: Algorithm Description

As introduced in the previous section, U-EDF is composed of two different phases. First, it uses the "horizontal" technique to give a budget of execution to every task on each processor (this phase is repeated whenever a new job arrives). Then, between two such budget pre-allocations (i.e., between two job releases), it schedules the active tasks according to their budgets, using a slight variation of EDF on each processor.

A. First phase: Pre-Allocation

As explained earlier, the first phase of our algorithm consists in pre-allocating time for the execution of tasks on processors, i.e., for each task determine the time budget reserved for its

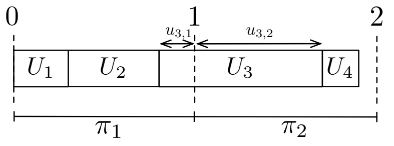


Fig. 2: Computation of $u_{i,j}$.

execution on each processor. Notice that we say "reserved" and not "performed". Indeed, as we reassign tasks at every new job arrival, there is no guaranty that we will actually execute τ_i for its allocated time on π_j.

Before explaining the allocation process, we first define some notations.

***Definition** 1 (current job allotment):* At any instant t, the allotment $\mathrm{al}_{i,j}(t)$ of task τ_i is the number of time units allocated on processor π_j to the execution of the *active job* of τ_i.

The allotment of τ_i on π_j is decreasing as τ_i is running on π_j. Hence, assuming that no new job arrives within $[t, t']$, if $\mathrm{exc}_{i,j}(t, t')$ denotes the time during which τ_i has been running on processor π_j in the interval $[t, t']$ then $\mathrm{al}_{i,j}(t') \overset{\text{def}}{=} \mathrm{al}_{i,j}(t) - \mathrm{exc}_{i,j}(t, t')$.

***Definition** 2 (future jobs reservation):* The future jobs reservation $\mathrm{res}_{i,j}(t_1, t_2)$ denotes the amount of time reserved on processor π_j for the execution of the *future jobs* of τ_i that might be released within $[t_1, t_2)$. This reservation is defined as:

$$\mathrm{res}_{i,j}(t_1, t_2) \overset{\text{def}}{=} (t_2 - t_1) \times u_{i,j}(t_1)$$

where $u_{i,j}(t) \overset{\text{def}}{=}$

$$\left[\sum_{\tau_x \in \mathrm{hp}_i(t) \cup \tau_i} U_x - (j-1)\right]_0^1 - \left[\sum_{\tau_x \in \mathrm{hp}_i(t)} U_x - (j-1)\right]_0^1$$

with $[x]_a^b \overset{\text{def}}{=} \max\{a, \min\{b, x\}\}$.

The $u_{i,j}(t)$ value denotes the proportion of the utilization factor of τ_i that will be reserved on processor π_j for the schedule of future jobs of τ_i. This value is obtained by aligning blocs of size U_i in an increasing current deadline $d_i(t)$ order, and then, by cutting this alignment in boxes of size 1 (see Figure 2). The proportion of U_i contained in the j^{th} box corresponds to $u_{i,j}(t)$. Finally, $\mathrm{res}_{i,j}(t_1, t_2)$ is equal to $u_{i,j}(t)$ multiplied by the length of the interval $[t_1, t_2)$. For instance, in Fig. 1(c), $\mathrm{res}_{1,1}(6, 10) = \frac{1}{3} \times (10 - 6) = \frac{4}{3}$ whereas $\mathrm{res}_{1,2}(6, 10) = 0$ because $u_{1,2} = 0$. Notice that this technique is inspired by McNaughton's algorithm presented in [22]. More details about the computation of this quantity are given in [1].

From this explanation on the computation of $\mathrm{res}_{i,j}(t_1, t_2)$, it is easy to see that the following equality holds:

***Property** 1:* If $U \leq m$, then it holds for every τ_i, t_1 and t_2 such that $t_2 > t_1$, that $\sum_{j=1}^{m} \mathrm{res}_{i,j}(t_1, t_2) = U_i \times (t_2 - t_1)$.

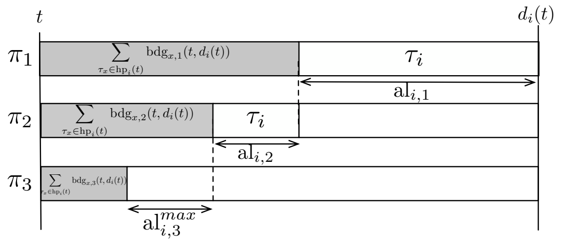

t
$d_i(t)$
π_1
$\sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{bdg}_{x,1}(t, d_i(t))$
τ_i
$\mathrm{al}_{i,1}$
π_2
$\sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{bdg}_{x,2}(t, d_i(t))$
τ_i
$\mathrm{al}_{i,2}$
π_3
$\sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{bdg}_{x,3}(t, d_i(t))$
$\mathrm{al}_{i,3}^{max}$

Fig. 3: Computation of $\mathrm{al}_{i,j}^{\max}(t)$ for τ_i on processor π_3.

Algorithm 1: Pre-allocation Algorithm.

```
Input:
TaskList := list of the $n$ tasks sorted by increasing absolute deadlines;
$t$ := current time;
forall the $\tau_i \in$ TaskList do
    for $j := 1$ to $m$ do
        $\mathrm{al}_{i,j}(t) := \min\{\mathrm{al}_{i,j}^{\max}(t)\ ,\ \mathrm{ret}_i(t) - \sum_{y<j} \mathrm{al}_{i,y}(t)\}$;
    end
end
```

Definition 3 (task budget): The budget $\mathrm{bdg}_{i,j}(t_1, t_2)$ denotes the total amount of time reserved for the task τ_i on processor π_j in the interval extending from t_1 to t_2 (with $t_2 \geq d_i(t_1)$). It includes the allotment $\mathrm{al}_{i,j}(t_1)$ for the currently active job of τ_i at time t_1 and the reservation $\mathrm{res}_{i,j}(d_i(t_1), t_2)$ for the execution of the future jobs of τ_i that might potentially arrive within $[d_i(t_1), t_2)$, i.e.,

$$\mathrm{bdg}_{i,j}(t_1, t_2) \overset{\mathrm{def}}{=} \mathrm{al}_{i,j}(t_1) + \mathrm{res}_{i,j}(d_i(t_1), t_2)$$

Notice that the two following properties can be derived from the previous definition.

Property 2: At any time t, the budget allocated to a task τ_i on processor π_j from time t to τ_i's deadline is equal to its allotment at time t, i.e., $\mathrm{bdg}_{i,j}(t, d_i(t)) = \mathrm{al}_{i,j}(t)$ (because $\mathrm{res}_{i,j}(d_i(t), d_i(t)) = 0$ from Definition 2).

Property 3: Let $t_1 < t_2 < t_3$ be three distinct instants such that $t_2 \geq d_i(t_1)$. It holds $\forall i, j$ that $\mathrm{bdg}_{i,j}(t_1, t_3) = \mathrm{bdg}_{i,j}(t_1, t_2) + \mathrm{res}_{i,j}(t_2, t_3)$.

Definition 4 (maximum allotment): At any time-instant t, the maximum allotment $\mathrm{al}_{i,j}^{\max}(t)$ of task τ_i on processor π_j is defined as the maximum number of time units that can be allocated on processor π_j to the active job of task τ_i in the time interval $[t, d_i(t)]$, i.e., $\mathrm{al}_{i,j}^{\max}(t)$ is an upper-bound on $\mathrm{al}_{i,j}(t)$. This is given by

$$\mathrm{al}_{i,j}^{\max}(t) \overset{\mathrm{def}}{=} \Big(d_i(t) - t\Big) - \sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{bdg}_{x,j}(t, d_i(t)) - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t)$$

The first term of this expression, i.e., $(d_i(t) - t)$, is the amount of time between instant t and the deadline of the active job of τ_i, the second term $\sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{bdg}_{x,j}(t, d_i(t))$ is the number of time units reserved in $[t, d_i(t)]$ for the tasks with a higher priority than τ_i, and the third term $\sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t)$ is the amount of time already reserved for the active job of τ_i on the processors of lower indices than π_j. This third term is subtracted in order to prevent the active job of τ_i from being executed concurrently on multiple processors (see Figure 3). Note that it can be shown that $\mathrm{al}_{i,j}^{\max}(t)$ is always non-negative.

Algorithm 1 presents the pre-allocation phase of U-EDF. Tasks are pre-allocated in an increasing current deadline order. Algorithm 1 maximizes the time allotted to each task τ_i on each processor. Hence, for every task τ_i, it assigns on each processor π_j the minimum between $\mathrm{al}_{i,j}^{\max}(t)$ and the amount of remaining execution time of τ_i that has not yet been assigned to other processors.

We easily recognize the "horizontal" approach presented in the previous section. Indeed, Algorithm 1 allocates time to tasks in an increasing current deadline order and maximizes the utilization of the first processors before starting using next ones.

B. Second phase: Schedule

Once the pre-allocation of the tasks amongst the processors has been done, we use an EDF-like algorithm named EDF-D (which stands for EDF with Delays) to schedule the tasks on the processors until the next job arrival (where a reallocation of the tasks will be performed again).

EDF-D is a scheduling algorithm which operates in the following way between two job allocations:

- It first builds the set $\mathcal{E}_j(t)$ of eligible tasks on π_j at time t. A task τ_i is eligible on π_j at time t if (i) it is not currently running on a processor of lower index; and (ii) τ_i still has to execute on processor π_j according to the current task allotment.
- At any time t, EDF-D schedules the *eligible* tasks 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 running on a processor π_j, it cannot be scheduled anymore on any processor with a higher index (see Figure 1(d) for an illustration). This leads to the following property:

Property 4: If a task τ_i with $\mathrm{al}_{i,j}(t) > 0$ is not running on processor π_j at time t, then either a task with a higher priority than τ_i is executing on π_j, or τ_i is running on a processor with a smaller index at time t.

C. U-EDF algorithm

Algorithm 2 summarizes the working process of U-EDF. Whenever an event occurs in the system (i.e., a task arrival, deadline or completion), Algorithm 2 is called. If the event in question is the release of a job, U-EDF recomputes a new task allocation amongst processors. Then, irrespective of the event type, EDF-D is used to 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 task allotment ;        // Algorithm 1
end
for $j := 1$ to $m$ do
    // Use EDF-D
    Update $\mathcal{E}_j(t)$ ;
    Execute on $\pi_j$ the task with the earliest deadline in $\mathcal{E}_j(t)$ ;
end
```

Note that U-EDF behaves exactly as EDF when the platform is composed of a single processor. Indeed, in this case, all tasks are obviously assigned to the same processor and a task cannot be delayed when using EDF-D as it never executes on another processor.

V. Optimality of U-EDF

In this section, we prove that U-EDF is optimal for the scheduling of sporadic tasks on multiprocessor, in the sense that it always meets all deadlines when $U \leq m$ and $U_i \leq 1$ for every $\tau_i \in \tau$. In these proofs, we assume that each job is executed for its worst-case execution time. However, because it is a slight variation of EDF which is used to schedule the tasks on each processor, if a job does not need to execute for its whole worst-case execution time, then all task deadlines are still respected. The jobs will just finish earlier than initially expected.

We starts by defining a valid assignment under U-EDF. Informally, an assignment is valid when U-EDF succeeds in entirely allocating all tasks on all processors using Algorithm 1. Formally,

***Definition** 5 (Valid U-EDF assignment):* An U-EDF assignment is said to be valid at time t iff the allotment of every task $\tau_i \in \tau$ is sufficient to successfully schedule its remaining execution time, i.e., $\forall \tau_i \in \tau : \sum_{j=1}^{m} \mathrm{al}_{i,j}(t) = \mathrm{ret}_i(t)$.

Since, according to Algorithm 2, all tasks have to be reassigned when a new job is released in the system, the proof of optimality (Theorem 2) is made by induction on the job arrival times. The induction is the following: assuming that U-EDF is valid at the arrival time t_r of job J_r, running EDF-D from t_r to the arrival of the next job J_{r+1} at time-instant t_{r+1}, we prove that (i) the U-EDF assignment is still valid at time-instant t_{r+1} (Lemmas 2 and 3) and (ii) all jobs with a deadline before or at t_{r+1} meet their deadline (Theorem 1).

***Theorem** 1 (from [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 $[t, t_\mathrm{a})$ then all jobs with a deadline such that $d_i(t) \leq t_\mathrm{a}$ meet their deadlines without intra-job parallelism.

Theorem 1 was proven in [1]. Notice that [1] deals with periodic tasks, but the proof of Theorem 1 was written for a more general model of tasks and still holds for the sporadic case.

***Lemma** 1:* Let t_0 be the very begining of the schedule, i.e., when no job has been released yet. For every $\tau_i \in \tau$ we have $\sum_{j=1}^{m} \mathrm{al}_{i,j}(t_0) = \mathrm{ret}_i(t_0)$

Proof: Since at time t_0 no job has been released yet, we have $\mathrm{ret}_i(t_0) = 0$ for every $\tau_i \in \tau$. Consequently, Algorithm 1 does not allocate any time unit for the execution of these tasks, i.e., $\mathrm{al}_{i,j}(t_0) = 0$, $\forall i, j$. Hence, it obviously holds that $\sum_{j=1}^{m} \mathrm{al}_{i,j}(t_0) = \mathrm{ret}_i(t_0)$.

■

Now that the basic case has been stated, we will prove through Lemmas 2 and 3 that the condition $\sum_{j=1}^{m} \mathrm{al}_{i,j}(t_\mathrm{a}) = \mathrm{ret}_i(t_\mathrm{a})$ is satisfied at any time $t_\mathrm{a} \geq t_0$ corresponding to a new job arrival.

We first define some new notations. Let J_a of task τ_a denote the job released at time t_a. According to Algorithm 2, all tasks have to be reassigned to the processors on J_a's release (i.e., at time t_a) using Algorithm 1. Since the set of currently active jobs is modified due to the release of J_a at time t_a, we distinguish between two instants t_a^- and t_a^+ defined as follows

- t_a^- considers the task set just *before* the release of J_a.
- t_a^+ considers the system right *after* the release of J_a and the execution of Algorithm 1.

It is essential to understand that from a theoretical point of view, these two instants are *equal to* t_a but correspond to two different states of the system.

Hence, Lemma 2 provides informations on the system until time t_a^-. Then, Lemma 3 proves that expression $\sum_{j=1}^{m} \mathrm{al}_{i,j}(t_\mathrm{a}) = \mathrm{ret}_i(t_\mathrm{a})$ is satisfied at the instant t_a^+.

***Lemma** 2:* Let t_a be the instant of the next arrival of a job after t and let t' be any time-instant such that $t < t' \leq t_\mathrm{a}^-$. If $\forall i, j$ we have $\mathrm{al}_{i,j}(t) \leq \mathrm{al}_{i,j}^{\max}(t)$ and tasks are scheduled using EDF-D in $[t, t']$, then it holds $\forall i, j$ that $\mathrm{al}_{i,j}(t') \leq \mathrm{al}_{i,j}^{\max}(t')$.

Proof: For all tasks τ_i that have completed their execution at time t', we have $\mathrm{ret}_i(t') = 0$ and $\mathrm{al}_{i,j}(t') = 0$ for all π_j. Consequently, the condition $\mathrm{al}_{i,j}(t') \leq \mathrm{al}_{i,j}^{\max}(t')$ is obviously satisfied for those tasks and the remainder of the proof considers only the tasks which have not completed their execution at time t', i.e., the tasks τ_i with $\mathrm{ret}_i(t') > 0$. Note that the active job of each such task τ_i cannot have a deadline $d_i(t)$ before time t'. Otherwise, since $\mathrm{ret}_i(t') > 0$, it would mean that τ_i has missed its deadline at time t', leading to a direct contradiction with Theorem 1. That is, we have $d_i(t) > t'$ and thus $d_i(t) = d_i(t')$. From this point onward, we will therefore use the notation d_i instead of $d_i(t)$ or $d_i(t')$ for the sake of conciseness.

Let us denote by $\mathrm{exc}_{i,j}(t, t')$ the time during which τ_i has been running on processor π_j in the time interval $[t, t']$. We have

$$\mathrm{al}_{i,j}(t) \overset{\text{def}}{=} \mathrm{al}_{i,j}(t') + \mathrm{exc}_{i,j}(t, t') \tag{1}$$

By assumption, we have $\forall \tau_i, \pi_j$ that $\mathrm{al}_{i,j}(t) \leq \mathrm{al}_{i,j}^{\max}(t)$ which gives, together with the above expression, $\mathrm{al}_{i,j}(t') + \mathrm{exc}_{i,j}(t,t') \leq \mathrm{al}_{i,j}^{\max}(t)$. Then, using Definitions 3 and 4, this expression can be rewritten as

$$\mathrm{al}_{i,j}(t') + \mathrm{exc}_{i,j}(t,t') \leq (d_i - t) - \sum_{\tau_x \in \mathrm{hp}_i(t)} \Big\{ \mathrm{al}_{x,j}(t) + \mathrm{res}_{x,j}(d_x, d_i) \Big\} - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t) \quad (2)$$

By replacing the term $\sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{al}_{x,j}(t)$ of Expression 2 with Expression 1, we get

$$\mathrm{al}_{i,j}(t') + \mathrm{exc}_{i,j}(t,t') \leq (d_i - t) - \sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{al}_{x,j}(t') - \sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{res}_{x,j}(d_x, d_i) - \sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{exc}_{x,j}(t,t') - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t)$$

Then, applying Definition 3 and subtracting $\mathrm{exc}_{i,j}(t,t')$ to both sides, we obtain

$$\mathrm{al}_{i,j}(t') \leq (d_i - t) - \sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{bdg}_{x,j}(t', d_i) - \sum_{\tau_x \in \mathrm{hp}_i(t) \cup \{\tau_i\}} \mathrm{exc}_{x,j}(t,t') - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t) \quad (3)$$

Let us now focus on the term $\sum_{\tau_x \in \mathrm{hp}_i(t) \cup \{\tau_i\}} \mathrm{exc}_{x,j}(t,t')$ in the above inequality. Intuitively, this term expresses the amount of time during which the active job of the tasks in $\mathrm{hp}_i(t) \cup \{\tau_i\}$ are executed on processor π_j in the interval $[t, t']$. Obviously, this amount of time cannot be greater than the time elapsed between instants t and t', i.e., we cannot use processor π_j during more than $(t' - t)$ time units within $[t, t']$. Hence, two cases may arise:

Case 1: $\sum_{\tau_x \in \mathrm{hp}_i(t) \cup \{\tau_i\}} \mathrm{exc}_{x,j}(t,t') = (t' - t)$

Note that the amount of time $\mathrm{al}_{i,y}(t)$ allotted to the active job of τ_i on a processor π_y decreases as τ_i executes on π_y implying that $\mathrm{al}_{i,y}(t') \leq \mathrm{al}_{i,y}(t)$ (remember that by assumption there is no reallocation of the tasks between t and t'), leading from Expression 3 to

$$\mathrm{al}_{i,j}(t') \leq (d_i - t') - \sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{bdg}_{x,j}(t', d_i) - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t')$$

and using Definition 4, we get $\mathrm{al}_{i,j}(t') \leq \mathrm{al}_{i,j}^{\max}(t')$.

Case 2: $\sum_{\tau_x \in \mathrm{hp}_i(t) \cup \{\tau_i\}} \mathrm{exc}_{x,j}(t,t') < (t' - t)$

In this case, there are some time units in $[t, t']$ for which processor π_j is either idle or it executes at least one task with a lower priority than τ_i (i.e., a task from the subset $\mathrm{lp}_i(t)$). There can be only two reasons for that to happen by using EDF-D (see Property 4):

a) The active job of τ_i at time t completes its execution on π_j during the time interval $[t, t']$. Hence $\mathrm{al}_{i,j}(t') = 0$ and it obviously holds that $\mathrm{al}_{i,j}(t') \leq \mathrm{al}_{i,j}^{\max}(t')$.

b) The execution of the active job of τ_i at time t was delayed on π_j, because in the time interval $[t, t']$ it was executed on at least one other processor π_y with $y < j$ (see Property 4). Specifically, the amount of time by which the execution of this active job is delayed on π_j is given by $\sum_{y=1}^{j-1} \mathrm{exc}_{i,y}(t,t')$ and it holds from Expression 1 that

$$\sum_{y=1}^{j-1} \mathrm{exc}_{i,y}(t,t') = \sum_{y=1}^{j-1} [\mathrm{al}_{i,y}(t) - \mathrm{al}_{i,y}(t')] \quad (4)$$

Furthermore, since τ_i did not finish its execution, the sum of all the execution times of all the tasks in $\mathrm{hp}_i(t) \cup \{\tau_i\}$, plus the delay incurred by τ_i must be greater than $(t' - t)$. That is,

$$\sum_{\tau_x \in \mathrm{hp}_i(t) \cup \{\tau_i\}} \mathrm{exc}_{x,j}(t,t') + \sum_{y=1}^{j-1} \mathrm{exc}_{i,y}(t,t') \geq (t' - t) \quad (5)$$

Using Expressions 3, 4 and 5 altogether, we finally obtain

$$\mathrm{al}_{i,j}(t') \leq (d_i - t') - \sum_{\tau_x \in \mathrm{hp}_i(t)} \mathrm{bdg}_{x,j}(t', d_i) - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t')$$

and using Definition 4, $\mathrm{al}_{i,j}(t') \leq \mathrm{al}_{i,j}^{\max}(t')$.

This states the lemma.

■

Lemma 3: Let $t \geq 0$ be any time-instant in the scheduling of τ and let $t_{\mathrm{a}} \geq t$ be the first instant after (or at) time t at which a job is released. If the assignment is valid at time t for every $\tau_i \in \tau$ and EDF-D is used to schedule tasks within $[t, t_{\mathrm{a}})$, then assuming that only one job is released at this instant t_{a}, we have for all tasks $\tau_i \in \tau : \sum_{j=1}^{m} \mathrm{al}_{i,j}(t_{\mathrm{a}}^{+}) = \mathrm{ret}_i(t_{\mathrm{a}}^{+})$ (i.e., there is a valid assignment after the reallocation of tasks at time t_a) provided that $\sum_{i=1}^{n} U_i \leq m$ and $U_i \leq 1$, $\forall \tau_i$.

Proof: Let J_a of task τ_a denote the (only) job released at time t_{a}. We already proved in Lemma 2, that at time t_{a}^{-} (i.e., just before the tasks reallocation), we have $\forall i, j$:

$$\mathrm{al}_{i,j}(t_{\mathrm{a}}^{-}) \leq \mathrm{al}_{i,j}^{\max}(t_{\mathrm{a}}^{-}) \quad (6)$$

Furthermore, the assignment was valid at time t and was not modified until t_{a}^{-}. Therefore, it must still be valid at time t_{a}^{-}. Hence, for every τ_i

$$\sum_{j=1}^{m} \mathrm{al}_{i,j}(t_{\mathrm{a}}^{-}) = \mathrm{ret}_i(t_{\mathrm{a}}^{-}) \quad (7)$$

Moreover, since by assumption, τ_a is the only task releasing a new job at time t_{a}, the deadlines and the remaining execution times of tasks $\tau_i \neq \tau_a$ do not change between t_{a}^{-} and t_{a}^{+}, i.e., $d_i(t_{\mathrm{a}}^{-}) = d_i(t_{\mathrm{a}}^{+})$ and $\mathrm{ret}_i(t_{\mathrm{a}}^{-}) = \mathrm{ret}_i(t_{\mathrm{a}}^{+})$. Hence, Expression 7 implies that $\sum_{j=1}^{m} \mathrm{al}_{i,j}(t_{\mathrm{a}}^{-}) =$

$\mathrm{ret}_i(t_\mathrm{a}^+)$ and applying Property 2, it holds that $\forall \tau_i \neq \tau_a$: $\sum_{j=1}^m \mathrm{bdg}_{i,j}(t_\mathrm{a}^-, d_i(t_\mathrm{a}^+)) = \mathrm{ret}_i(t_\mathrm{a}^+)$.

On the other hand, after the arrival of its new job, task τ_a has a new deadline $d_a(t_\mathrm{a}^+)$ equal to $t_\mathrm{a} + D_a$, and its remaining execution time is now equal to its worst-case execution time, i.e., $\mathrm{ret}_a(t_\mathrm{a}^+) = C_a$, and because τ_a already reached its deadline (from Theorem 1), it must hold that $\mathrm{al}_{a,j}(t_\mathrm{a}^-) = 0$. Therefore, from Definition 3 $\sum_{j=1}^m \mathrm{bdg}_{a,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) = \sum_{j=1}^m \mathrm{res}_{a,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+))$ and applying Property 1, it holds that $\sum_{j=1}^m \mathrm{bdg}_{a,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) = U_a \times D_a = C_a = \mathrm{ret}_a(t_\mathrm{a}^+)$. Hence, for all tasks τ_i in τ, we have

$$\sum_{j=1}^m \mathrm{bdg}_{i,j}(t_\mathrm{a}^-, d_i(t_\mathrm{a}^+)) = \mathrm{ret}_i(t_\mathrm{a}^+) \tag{8}$$

For the sake of readability, we will assume from this point onward that tasks are indexed according to their current deadlines at time t_a^+. Hence, if $r < s$ then $d_r(t_\mathrm{a}^+) \leq d_s(t_\mathrm{a}^+)$, implying that τ_r has a higher priority than τ_s at time t_a^+.

The remainder of the proof is subdivided in two subproofs. In Subproof 1, we show that, $\forall i, j$

$$\sum_{p=1}^j \sum_{q=1}^i \mathrm{bdg}_{q,p}(t_\mathrm{a}^+, d_i(t_\mathrm{a})) \geq \sum_{p=1}^j \sum_{q=1}^i \mathrm{bdg}_{q,p}(t_\mathrm{a}^-, d_i(t_\mathrm{a}))$$

Then, in Subproof 2, we deduce from this expression that $\sum_{j=1}^m \mathrm{al}_{i,j}(t_\mathrm{a}^+) = \mathrm{ret}_i(t_\mathrm{a}^+)$ for every task τ_i in τ, thereby proving the lemma.

Subproof 1: The first subproof is made by induction.
Let us assume that for task τ_{i-1}, we have for all processors $k = \{1, ..., m\}$:

$$\sum_{p=1}^k \sum_{q=1}^{i-1} \mathrm{bdg}_{q,p}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+)) \geq \sum_{p=1}^k \sum_{q=1}^{i-1} \mathrm{bdg}_{q,p}(t_\mathrm{a}^-, d_i(t_\mathrm{a}^+)) \tag{9}$$

We prove for task τ_i that

$$\sum_{p=1}^k \sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+)) \geq \sum_{p=1}^k \sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_\mathrm{a}^-, d_i(t_\mathrm{a}^+)) \tag{10}$$

Basic statement: Notice that the induction hypothesis (i.e., Expression 9) is obviously respected when $i = 1$ since both sides of Expression 9 are then equal to 0.

Induction Step: Algorithm 1 maximizes the allotment of τ_i on π_j at time t_a^+. Hence, according to line 3 of Algorithm 1, we either have $\sum_{p=1}^j \mathrm{al}_{i,p}(t_\mathrm{a}^+) = \mathrm{ret}_i(t_\mathrm{a}^+)$ or $\mathrm{al}_{i,j}(t_\mathrm{a}^+) = \mathrm{al}_{i,j}^{\max}(t_\mathrm{a}^+)$:

Case 1. If $\sum_{p=1}^j \mathrm{al}_{i,p}(t_\mathrm{a}^+) = \mathrm{ret}_i(t_\mathrm{a}^+)$, then, it means that τ_i has been entirely allocated on processors π_1 to π_j. Since, according to Expression 8, τ_i has a budget of execution of exactly $\mathrm{ret}_i(t_\mathrm{a}^+)$ time units dispersed among the processors at time t_a^-, it must hold that $\sum_{p=1}^j \mathrm{al}_{i,p}(t_\mathrm{a}^+) = \sum_{p=1}^j \mathrm{bdg}_{i,p}(t_\mathrm{a}^-, d_i(t_\mathrm{a}^+))$. Using Property 2, we get

$$\sum_{p=1}^j \mathrm{bdg}_{i,p}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+)) = \sum_{p=1}^j \mathrm{bdg}_{i,p}(t_\mathrm{a}^-, d_i(t_\mathrm{a}^+))$$

The induction hypothesis (Expression 9) gives for $k = j$,

$$\sum_{p=1}^j \sum_{q=1}^{i-1} \mathrm{bdg}_{q,p}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+)) \geq \sum_{p=1}^j \sum_{q=1}^{i-1} \mathrm{bdg}_{q,p}(t_\mathrm{a}^-, d_i(t_\mathrm{a}^+))$$

Therefore, combining the two previous expressions leads to

$$\sum_{p=1}^j \sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+)) \geq \sum_{p=1}^j \sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_\mathrm{a}^-, d_i(t_\mathrm{a}^+))$$

which proves Expression 10.

Case 2. If $\mathrm{al}_{i,j}(t_\mathrm{a}^+) = \mathrm{al}_{i,j}^{\max}(t)$ then it holds from Definition 4 that

$$\mathrm{al}_{i,j}(t_\mathrm{a}^+) = (d_i(t_\mathrm{a}^+) - t_\mathrm{a}) - \sum_{x=1}^{i-1} \mathrm{bdg}_{x,j}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+)) - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t_\mathrm{a}^+)$$

Because $\mathrm{al}_{i,j}(t_\mathrm{a}^+) = \mathrm{bdg}_{i,j}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+))$ by Property 2 and adding $\sum_{x=1}^{i-1} \mathrm{bdg}_{x,j}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+))$ to both sides, we get

$$\sum_{k=1}^{i} \mathrm{bdg}_{k,j}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+)) = (d_i(t_\mathrm{a}^+) - t_\mathrm{a}) - \sum_{y=1}^{j-1} \mathrm{bdg}_{i,y}(t_\mathrm{a}^+, d_i(t_\mathrm{a}^+)) \tag{11}$$

Two different cases should now be analyzed: $\tau_i \neq \tau_a$ or $\tau_i = \tau_a$. However, due to space limitations, we only give the proof for the first case (i.e., for $\tau_i \neq \tau_a$). The proof for $\tau_i = \tau_a$ is almost identical and can be consulted in the appendix.

Note that for every task τ_i which does not release a job at time t_a (i.e., every $\tau_i \neq \tau_a$), we have $d_i(t_\mathrm{a}^-) = d_i(t_\mathrm{a}^+)$ and $\mathrm{ret}_i(t_\mathrm{a}^-) = \mathrm{ret}_i(t_\mathrm{a}^+)$. Therefore, in the remainder of this subproof, we will use the notations $d_i(t_\mathrm{a})$ and $\mathrm{ret}_i(t_\mathrm{a})$ to refer to the deadline and the remaining execution time of task τ_i at both instants t_a^- and t_a^+.

Hence, using Definition 4, we have at time t_a^-,

$$\mathrm{al}_{i,j}^{\max}(t_\mathrm{a}^-) = (d_i(t_\mathrm{a}) - t_\mathrm{a}) - \sum_{x=1}^{i-1} \mathrm{bdg}_{x,j}(t_\mathrm{a}^-, d_i(t_\mathrm{a})) - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t_\mathrm{a}^-)$$

Expression 6 yields

$$\mathrm{al}_{i,j}(t_\mathrm{a}^-) \leq (d_i(t_\mathrm{a}) - t_\mathrm{a}) - \sum_{x=1}^{i-1} \mathrm{bdg}_{x,j}(t_\mathrm{a}^-, d_i(t_\mathrm{a})) - \sum_{y=1}^{j-1} \mathrm{al}_{i,y}(t_\mathrm{a}^-)$$

and because $\mathrm{al}_{i,j}(t_\mathrm{a}) = \mathrm{bdg}_{i,j}(t_\mathrm{a}, d_i(t_\mathrm{a}))$ (from Property 2), adding $\sum_{x=1}^{i-1} \mathrm{bdg}_{x,j}(t_\mathrm{a}^-, d_i(t_\mathrm{a}))$ to both sides gives

$$\sum_{k=1}^{i} \mathrm{bdg}_{k,j}(t_\mathrm{a}^-, d_i(t_\mathrm{a})) \leq (d_i(t_\mathrm{a}) - t_\mathrm{a}) - \sum_{y=1}^{j-1} \mathrm{bdg}_{i,y}(t_\mathrm{a}^-, d_i(t_\mathrm{a})) \tag{12}$$

Then, combining Expressions 11 and 12 leads to

$$\sum_{k=1}^{i} \mathrm{bdg}_{k,j}(t_\mathrm{a}^+, d_i(t_\mathrm{a})) + \sum_{y=1}^{j-1} \mathrm{bdg}_{i,y}(t_\mathrm{a}^+, d_i(t_\mathrm{a})) \geq$$
$$\sum_{k=1}^{i} \mathrm{bdg}_{k,j}(t_\mathrm{a}^-, d_i(t_\mathrm{a})) + \sum_{y=1}^{j-1} \mathrm{bdg}_{i,y}(t_\mathrm{a}^-, d_i(t_\mathrm{a}))$$

Moreover, the induction hypothesis (Eq. 9) gives for $k = j-1$

$$\sum_{p=1}^{j-1}\sum_{q=1}^{i-1} \mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{+}, d_i(t_{\mathrm{a}})) \geq \sum_{p=1}^{j-1}\sum_{q=1}^{i-1} \mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{-}, d_i(t_{\mathrm{a}}))$$

Finally, the previous two expressions lead to

$$\sum_{p=1}^{j}\sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{+}, d_i(t_{\mathrm{a}})) \geq \sum_{p=1}^{j}\sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{-}, d_i(t_{\mathrm{a}}))$$

which proves Expression 10.

Subproof 2. We now prove that $\sum_{p=1}^{m} \mathrm{al}_{q,p}(t_{\mathrm{a}}^{+}) = \mathrm{ret}_q(t_{\mathrm{a}}^{+})$ for all tasks τ_q.

Using Expression 10, proved in Subproof 1, we have for processor π_m

$$\sum_{p=1}^{m}\sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{+}, d_i(t_{\mathrm{a}}^{+})) \geq \sum_{p=1}^{m}\sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{-}, d_i(t_{\mathrm{a}}^{+})) \quad (13)$$

Using Property 3 on both sides of Expression 13, we get

$$\sum_{p=1}^{m}\sum_{q=1}^{i} \left[\mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{+}, d_q(t_{\mathrm{a}}^{+})) + \mathrm{res}_{q,p}(d_q(t_{\mathrm{a}}^{+}), d_i(t_{\mathrm{a}}^{+}))\right] \geq$$
$$\sum_{p=1}^{m}\sum_{q=1}^{i} \left[\mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{-}, d_q(t_{\mathrm{a}}^{+})) + \mathrm{res}_{q,p}(d_q(t_{\mathrm{a}}^{+}), d_i(t_{\mathrm{a}}^{+}))\right]$$

implying

$$\sum_{p=1}^{m}\sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{+}, d_q(t_{\mathrm{a}}^{+})) \geq \sum_{p=1}^{m}\sum_{q=1}^{i} \mathrm{bdg}_{q,p}(t_{\mathrm{a}}^{-}, d_q(t_{\mathrm{a}}^{+}))$$

Finally, applying Definition 3 and Expression 8, this leads to

$$\sum_{q=1}^{i}\sum_{p=1}^{m} \mathrm{al}_{q,p}(t_{\mathrm{a}}^{+}) \geq \sum_{q=1}^{i} \mathrm{ret}_q(t_{\mathrm{a}}^{+})$$

Since Algorithm 1 never allocates more than $\mathrm{ret}_q(t_{\mathrm{a}}^{+})$ to a task τ_q at time t_{a}^{+} (see line 3 of Algorithm 1), the previous inequality is true only if, $\forall q : \sum_{p=1}^{m} \mathrm{al}_{q,p}(t_{\mathrm{a}}^{+}) = \mathrm{ret}_q(t_{\mathrm{a}}^{+})$.

This proves the lemma.

■

***Theorem** 2:* U-EDF is optimal for the schedule of sporadic tasks with implicit deadlines such that $\sum_{i=1}^{n} U_i \leq m$ and $U_i \leq 1$ for every task $\tau_i \in \tau$.

Proof: By Lemma 1, we know that there is a valid U-EDF assignment at time $t = t_0$ assuming that no job has been released in the system, yet. Next, assuming that $\sum_{i=1}^{n} U_i \leq m$ and $U_i \leq 1, \forall \tau_i \in \tau$, Lemma 3 shows that, if EDF-D is used to schedule the tasks between two consecutive task allocations, then a valid U-EDF assignment exists at any instant corresponding to a new job arrival (Notice that, if t_{a} is the arrival time of the new job, t_{a} can be equal to t. Therefore, if two or more jobs arrive at the same instant t_{a}, we just have to apply multiple times Lemma 3).

Furthermore, Theorem 1 proves that all deadlines between two such job arrivals are met. Notice that, if no new job arrives after a given time instant t, Theorem 1 ensures that all tasks meet their deadlines by assuming that the next job arrival is $t_{\mathrm{a}} = +\infty$. Therefore, U-EDF is optimal for the schedule of sporadic tasks with implicit deadlines. ■

VI. Simulation Results

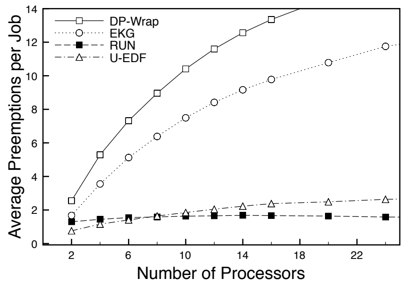
We evaluated the performance of U-EDF through extensive simulations. In each experiment, we simulated the scheduling of 1,000 task sets from time 0 to time 100,000. Each task had a minimum inter-arrival time randomly chosen within $[5, 100]$ using a uniform integer distribution. Task utilizations were randomly generated between 0.01 and 0.99 until the targeted system utilization was reached.

Both the clustering and the virtual processing techniques presented in [1] were implemented for U-EDF. The interested reader can refer to [1] for further details.

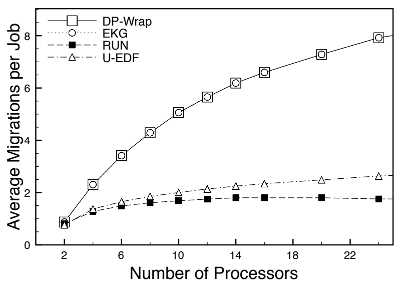
For Figures 4(a) and 4(b), we simulated the scheduling of implicit deadlines periodic tasks with a total utilization of 100% of the platform on a varying number of processors. We compared the average number of preemptions and migrations generated by U-EDF with that of the three most efficient *optimal* scheduling algorithms for *periodic* tasks on multiprocessor, namely DP-Wrap, EKG and RUN. Both RUN and U-EDF clearly outperform EKG and DP-Wrap. This result can be explained by the absence of fairness in the schedule produced by RUN and U-EDF. Furthermore, U-EDF shows to be slightly better than RUN for a number of processors smaller than 8. Hence, U-EDF seems to be the best alternative to use on platforms composed of clusters containing less than 8 processors. Moreover, unlike RUN which cannot schedule sporadic tasks on multiprocessor platforms, U-EDF is easily extendable to the scheduling of more general task models. For now, U-EDF is the only existing optimal algorithm for the scheduling of *sporadic* tasks which is not based on the notion of fairness.

Figure 4(c) presents two different results on a platform composed of 8 processors with a total utilization varying between 5 and 100%. First, we compare the average number of preemptions per job generated by U-EDF and Partitioned-EDF (P-EDF)[1] when tasks are periodic. Then, we show the results obtained for the scheduling of sporadic tasks. For each sporadic task we randomly selected the maximum delay that jobs can incur, in the range $[1, 100]$. Then, each job release was delayed by a random number of time units uniformly generated between 0 and this maximum delay. Note that for P-EDF results, only task sets for which all task deadlines were met are taken into account in the calculation of the number of preemptions and migrations. We can see on Figure 4(c) that U-EDF and P-EDF have the same average number of preemptions when the system utilization does not exceed 50%. This particularity is the result of the clustering technique explained in [1]. The second information we can draw from these graphs is that the average number of preemptions generated by U-EDF is slightly higher for sporadic tasks. This is due to the fact that even if there are

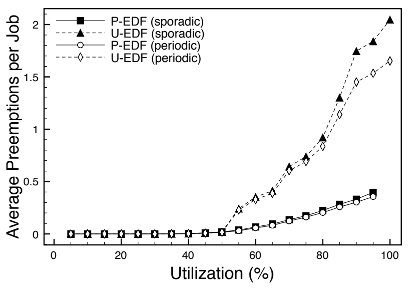
[1]Tasks are partitioned with the worst fit decreasing utilization heuristic



(a)



(b)



(c)

Fig. 4: (a) and (b) periodic tasks running on a number of processors varying between 2 and 24 with a total utilization of 100%; (c) periodic and sporadic tasks running on 8 processors with a total utilization within 5 and 100%.

few active tasks at some time t, the pre-allocation phase assigns these tasks on as few processors as possible. Hence there are more tasks interacting with each other than with P-EDF. Nevertheless, in its current version, U-EDF is not work conserving (i.e., a processor might stay idle, even if there is a task with remaining execution time which is not running on the platform). As a consequence, by using the processors idle times (which are more likely in sporadic systems) to schedule pending tasks in a least remaining execution time order for instance, we could certainly further reduce the average number of preemptions incurred by jobs under U-EDF, and have similar results than P-EDF. Note that even though the U-EDF curves deviate from the P-EDF ones on Figure 4(c) for a system utilization greater than 50%, unlike U-EDF, Partitioned-EDF is not optimal and can miss task deadlines.

A Note on the Implementation: It can easily be shown that the pre-allocation algorithm has a run-time complexity of $O(n \times m)$ which may seem excessive as Algorithm 1 is invoked at each job release. However, the processing platforms include more and more processors/cores every day. For instance, Tilera designed processors with 36 to 100 integrated cores [23]. In these conditions, it seems quite realistic to dedicate one core to the execution of a complex scheduling algorithm in order to drastically increase the total utilization of the platform when comparing to simpler algorithms, while keeping a small number of preemptions and migrations.

VII. Conclusion and Future Works

In this paper, we presented the first *optimal* algorithm for the scheduling of *sporadic* tasks on *multiprocessor* which does not fairly distribute the platform processing capacity amongst tasks (or group of tasks). This new technique has shown to be efficient in terms of preemptions and migrations. Moreover, U-EDF seems to be easily extendable to the scheduling of more general model of tasks and systems. Hence, multi-threaded parallel tasks could be scheduled efficiently with U-EDF on multiprocessor platforms adopting the methodology proposed in [24]. Similarly, we are considering generalizing U-EDF to the scheduling of dynamic task systems. Finally, we believe that U-EDF could still be optimized to further reduce the number of preemptions, migrations and scheduling points.

Acknowledgements

This work was supported by the Belgian National Science Foundation (FNRS.) under a FRIA grant, and by the RECOMP project, ref. ARTEMIS/0202/2009, co-funded by FCT-MCTES (Portuguese Foundation for Science and Technology), and by EU funds through the ARTEMIS Joint Undertaking, under grant nr. 100202.

References

[1] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, “Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness,” in *RTCSA’11*, 2011, pp. 15–24.

[2] S. K. Baruah and T. P. Baker, “Schedulability analysis of global EDF,” *Real-Time Systems*, vol. 38, no. 3, pp. 223–235, 2008.

[3] S. Kato and N. Yamasaki, “Portioned EDF-based scheduling on multiprocessors,” in *EMSOFT ’08*, 2008, pp. 139–148.

[4] S. Kato, N. Yamasaki, and Y. Ishikawa, “Semi-partitioned scheduling of sporadic task systems on multiprocessors,” in *ECRTS’ 09*, 2009, pp. 249–258.

[5] A. Burns, R. Davis, P. Wang, and F. Zhang, “Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme,” *Real-Time Systems*, vol. 48, no. 1, pp. 3–33, 2011.

[6] A. Srinivasan and J. H. Anderson, “Optimal rate-based scheduling on multiprocessors,” in *STOC ’02*, 2002, pp. 189–198.

[7] J. H. Anderson and A. Srinivasan, “Early-release fair scheduling,” in *ECRTS ’00*, 2000, pp. 35–43.

[8] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, “DP-Fair: a unifying theory for optimal hard real-time multiprocessor scheduling,” *Real-Time Systems*, vol. 47, pp. 389–429, 2011.

[9] H. Cho, B. Ravindran, and E. D. Jensen, “An optimal real-time scheduling algorithm for multiprocessors,” in *RTSS ’06*, 2006, pp. 101–110.

[10] D. Zhu *et al.*, “An optimal boundary fair scheduling algorithm for multiprocessor real-time systems,” *Journal of Parallel and Distributed Computing*, vol. 71, no. 10, pp. 1411–1425, 2011.

[11] B. Andersson and E. Tovar, “Multiprocessor scheduling with few preemptions,” in *RTCSA ’06*, 2006, pp. 322–334.

[12] 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*, 1993, pp. 345–354.

[13] ——, “Proportionate progress: A notion of fairness in resource allocation,” *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[14] S. K. Baruah, J. Gehrke, and C. G. Plaxton, “Fast scheduling of periodic tasks on multiple resources,” in *IPPS ’95*, 1995, pp. 280–288.

[15] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, “RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor,” in *RTSS ’11*, December 2011, pp. 104–115.

[16] S. Funk and V. Nadadur, “LRE-TL: An optimal multiprocessor algorithm for sporadic task sets,” in *RTNS’ 09*, October 2009, pp. 159–168.

[17] B. Andersson and K. Bletsas, “Sporadic multiprocessor scheduling with few preemptions,” in *ECRTS’08*, 2008, pp. 243–252.

[18] K. Bletsas and B. Andersson, “Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound,” *Real-Time Systems*, vol. 47, pp. 319–355, 2011.

[19] A. Bastoni, B. Brandenburg, and J. Anderson, “Is Semi-Partitioned Scheduling Practical?” in *ECRTS’11*, 2011, pp. 125–135.

[20] 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.

[21] A. Khemka and R. Shyamasundar, “An optimal multiprocessor real-time scheduling algorithm,” *Journal of Parallel and Distributed Computing*, vol. 43, no. 1, pp. 37–45, May 1997.

[22] R. McNaughton, “Scheduling with deadlines and loss functions,” *Management Science*, vol. 6, no. 1, pp. 1–12, 1959.

[23] Tilera, “Tile-gx 3000 series overview,” 2011. [Online]. Available: http://www.tilera.com/sites/default/files/productbriefs/TILE-Gx 3000 Series Brief.pdf

[24] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, “Optimizing the number of processors to schedule multi-threaded tasks,” in *RTSS’11-WiP Session*, December 2011, pp. 5–8.

APPENDIX

A. Additional Proof Sketch of Lemma 3

This lemma has already been partially proven in Section V. Hence, in this appendix, we only prove the missing part, i.e., we prove Expression 10 with $\tau_i = \tau_a$ when we are in case 2 of Subproof 1 in Lemma 3. That is, we demonstrate that for all $j \in \{1, ..., m\}$

$$\sum_{p=1}^{j} \sum_{\tau_q \in \mathrm{hp}_a(t_\mathrm{a}^+) \cup \tau_a} \mathrm{bdg}_{q,p}(t_\mathrm{a}^+, d_a(t_\mathrm{a}^+)) \geq \sum_{p=1}^{j} \sum_{\tau_q \in \mathrm{hp}_a(t_\mathrm{a}^+) \cup \tau_a} \mathrm{bdg}_{q,p}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) \tag{14}$$

assuming that τ_a is the (only) task releasing a job at time t_a and $\mathrm{al}_{a,j}(t_\mathrm{a}^+) = \mathrm{al}_{a,j}^{\max}(t_\mathrm{a}^+)$.

The difficulty for this particular task is that $\mathrm{hp}_a(t_\mathrm{a}^+)$ might contain more tasks than $\mathrm{hp}_a(t_\mathrm{a}^-)$. Indeed, the current deadline of τ_a which was equal to t_a before the arrival of its new job (see definition of the task current deadline in Section II) is now equal to the deadline of its newly arrived job implying that $d_a(t_\mathrm{a}^-) < d_a(t_\mathrm{a}^+)$. Hence, there may exist some tasks τ_i such that $d_i(t_\mathrm{a}) \geq d_a(t_\mathrm{a}^-)$ and $d_i(t_\mathrm{a}) < d_a(t_\mathrm{a}^+)$ leading to $\tau_i \notin \mathrm{hp}_a(t_\mathrm{a}^-)$ and $\tau_i \in \mathrm{hp}_a(t_\mathrm{a}^+)$ (we remind that the current deadline of every task different from τ_a does not change).

Two cases could arise considering the value of $\mathrm{bdg}_{a,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+))$ for $j \in \{1, ..., m\}$:

- $\mathrm{bdg}_{a,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) \leq (d_a(t_\mathrm{a}^+) - t_\mathrm{a}) - \sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+)} \mathrm{bdg}_{x,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) - \sum_{y=1}^{j-1} \mathrm{bdg}_{a,y}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+))$:
  In this case we can apply the same argument than for $\tau_i \neq \tau_a$ (see proof of Lemma 3 in Section V). That is, we first combine the above expression with Expressions 11. Then, using the induction hypothesis (Expression 9) for $k = j - 1$, we prove Expression 14.
- $\mathrm{bdg}_{a,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) > (d_a(t_\mathrm{a}^+) - t_\mathrm{a}) - \sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+)} \mathrm{bdg}_{x,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) - \sum_{y=1}^{j-1} \mathrm{bdg}_{a,y}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+))$:
  We are in the situation depicted on Figure 5(a). Let Δ be the difference between the left side and the right side of this inequality. That is,

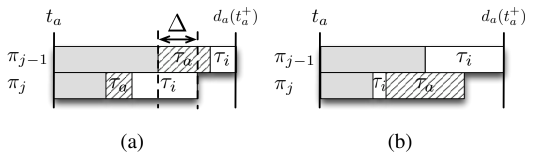


Fig. 5: Situation when $\tau_i \notin \mathrm{hp}_a(t_\mathrm{a}^-)$ but $\tau_i \in \mathrm{hp}_a(t_\mathrm{a}^+)$ (a) before re-allocation of tasks; (b) after re-allocation of tasks.

$$\Delta = \mathrm{bdg}_{a,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) - (d_a(t_\mathrm{a}^+) - t_\mathrm{a}) + \sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+)} \mathrm{bdg}_{x,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) + \sum_{y=1}^{j-1} \mathrm{bdg}_{a,y}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) \tag{15}$$

Since Algorithm 1 maximizes the amount of time allocated to any task τ_i on processors with smallest indices and because at time t_a Algorithm 1 allocates time to the tasks $\tau_i \in \mathrm{hp}_a(t_\mathrm{a}^+)$ first, the budget of time that was reserved for the task τ_a on processors π_1 to π_{j-1} will now preferably be assigned to the tasks $\tau_i \in \mathrm{hp}_a(t_\mathrm{a}^+)$ (see Figure 5(b)). Therefore, it can easily be shown that the budget of time of the tasks in $\mathrm{hp}_a(t_\mathrm{a}^+)$ on the processors π_1 to π_{j-1} will increase by at least Δ time units and the budget of time of τ_a on the processors π_1 to π_{j-1} will decrease by at least Δ time units (see Figure 5(b)). Therefore, using Definition 4, we have on processor π_j

$$\begin{aligned}
\mathrm{al}_{a,j}^{\max}(t_\mathrm{a}^+) &= (d_a(t_\mathrm{a}^+) - t_\mathrm{a}) - \sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+)} \mathrm{bdg}_{x,j}(t_\mathrm{a}^+, d_a(t_\mathrm{a}^+)) \\
&\quad - \sum_{y=1}^{j-1} \mathrm{al}_{a,y}(t_\mathrm{a}^+) \\
&\geq (d_a(t_\mathrm{a}^+) - t_\mathrm{a}) - \sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+)} \mathrm{bdg}_{x,j}(t_\mathrm{a}^+, d_a(t_\mathrm{a}^+)) \\
&\quad - \sum_{y=1}^{j-1} \mathrm{bdg}_{a,j-1}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) + \Delta
\end{aligned}$$

Then, replacing Δ by its expression given in Expression 15 and rearranging the terms, we get

$$\mathrm{al}_{a,j}^{\max}(t_\mathrm{a}^+) + \sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+)} \mathrm{bdg}_{x,j}(t_\mathrm{a}^+, d_a(t_\mathrm{a}^+)) \geq \mathrm{bdg}_{a,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+)) + \sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+)} \mathrm{bdg}_{x,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+))$$

Because $\mathrm{al}_{a,j}(t_\mathrm{a}^+) = \mathrm{al}_{a,j}^{\max}(t_\mathrm{a}^+)$ by assumption, applying Property 2 we get

$$\sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+) \cup \tau_a} \mathrm{bdg}_{x,j}(t_\mathrm{a}^+, d_a(t_\mathrm{a}^+)) \geq \sum_{\tau_x \in \mathrm{hp}_a(t_\mathrm{a}^+) \cup \tau_a} \mathrm{bdg}_{x,j}(t_\mathrm{a}^-, d_a(t_\mathrm{a}^+))$$

and using the induction hypothesis (Expression 9) with $k = j - 1$, we prove Expression 14.