Swapping to Reduce Preemptions and Migrations in EKG

Geoffrey Nelissen[†1], Shelby Funk[§], Joël Goossens[†], Dragomir Milojevic[†]

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

[§]Department of Computer Science
University of Georgia
Athens, GA, USA

***Abstract*—EKG is a multiprocessor scheduling algorithm which is optimal for the schedule of real-time periodic tasks with implicit deadlines. It adheres to the deadline partitioning fair (DP-Fair) approach. However, it was shown in recent studies that the systematic execution of some tasks inherent in such approaches, significantly reduce the usability of this algorithm. Hence, we propose a swapping algorithm with the aim of reducing the number of preemptions and migrations incurred by EKG. This algorithm should enhance the practicality of EKG while keeping its optimality.**

I. Introduction

Over the last two decades, numerous optimal multiprocessor scheduling algorithms for periodic tasks have been proposed [1]–[7]. Each new result attempts to outperform previous algorithms in terms of preemptions, migrations, number of scheduling points or time complexity. Indeed, many studies state that the run-time overheads caused by these various factors, dramatically impact the usability of optimal algorithms in real applications [8]–[10].

Recently, Levin *et al.* developed a *deadline partitioning fair* (DP-Fair) theory explaining how the optimality could be reached on multiprocessor platforms by ensuring the *fairness* for all tasks at the deadlines of jobs released in the system [4]. In this approach, the time is divided in *time slices*. All tasks are assigned a *local execution time* in each time slice, which is determined so as to ensure that all deadlines will be met.

In 2006, Andersson *et al.* developed the algorithm EKG which categorizes tasks as *migratory* or *non-migratory* [5]. Migratory tasks are scheduled according to the DP-Fair approach, while non-migratory tasks are scheduled under an EDF scheduling policy. The EKG algorithm seems very promising in terms of reducing preemptions and migrations. However, recent studies showed that the systematic schedule of migratory tasks in each time slice significantly increases the number of preemptions and migrations and hence has a negative impact on the schedulability of task systems on real computational platforms [10].

We propose a technique to address the preemption and migration overheads in EKG. We present a swapping algorithm which increases (or decreases) the time reserved for migratory tasks in each time slice, thereby, suppressing migratory tasks (and associated run-time overheads) from time slices where their execution is not required to keep a correct schedule.

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

II. Model

We consider the scheduling of n periodic tasks with implicit deadlines on m identical processors. Each task τ_i in the set τ is characterized by a period T_i and a worst case execution time C_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 denote the utilization 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 met its deadline.

Since we are working with implicit deadline tasks, each task has one active job at any time t. We can therefore write without ambiguity that the deadline $d_i(t)$ of the task τ_i at time t is the deadline of the current active job of τ_i at time t. Similarly, we denote the remaining execution time of the current active job of τ_i at time t as $r_i(t)$.

In the remainder of this paper, we will assume that, at any instant t, the set τ is ordered according to the deadlines of the tasks at time t. That is, for any two tasks τ_x and τ_y, if $x < y$ then $d_x(t) \leq d_y(t)$.

III. Overview of the EKG Algorithm

EKG, first proposed by Andersson and Tovar in [5], is the short-hand notation for ***EDF with task splitting and k processors in a Group***. As indicated by its name, it divides the computational platform into clusters through the definition of a parameter k. In each cluster, we have $k \leq m$ processors. A bin-packing algorithm is used to partition the tasks among the clusters so that the total utilization on each cluster is not greater than k. Also, in order to minimize the number of preemptions and migrations, every task τ_i with a utilization U_i greater than $\frac{k}{k+1}$ receives its own processor.

It was proved in [5] that EKG ensures a utilization bound of $(\frac{k}{k+1} \cdot m)$ when $k < m$, and is optimal for the schedule of periodic tasks with implicit deadlines when $k = m$.

In the remainder of this paper, we will assume that there is only one cluster in the system. However, if there should be multiple clusters, the same reasoning could be applied on each individual cluster without any variation.

After the partitioning of τ onto the clusters, EKG works in two different phases. First, it assigns the tasks in each cluster among the processors. Then, it schedules the tasks in accordance with this assignment.

The assignment follows a *first fit* heuristic. That is, tasks are assigned to a processor as long as the capacity c_j on this processor is not exhausted. Let $u_{i,j}$ denote the proportion of

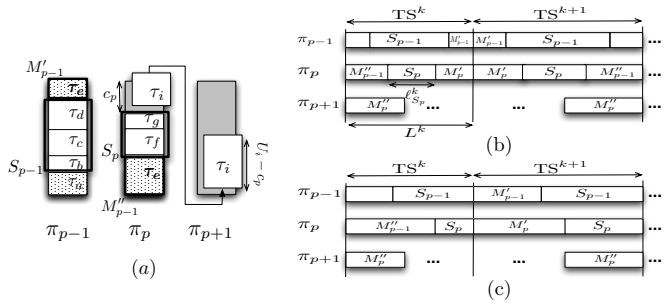

Fig. 1. (a) Assignment, (b) schedule before and (c) after swap under EKG.

τ_i's utilization assigned on processor π_j. We define S_j as the set of tasks entirely assigned on π_j. That is,

$$S_j \stackrel{\text{def}}{=} \{\tau_i \in \tau \mid u_{i,j} = U_i\}.$$

If we assume that τ_i is the next task to assign, and π_p is the first processor with some remaining capacity (i.e., $c_p > 0$ and $c_j = 0$ for all $j < p$), if $c_p \geq U_i$ then $S_p = S_p \cup \{\tau_i\}$. Otherwise, τ_i becomes a migratory task, denoted M_p, which is split into two subtasks M'_p with a utilization factor $U_{M'_p} = c_p$, and M''_p with a utilization $U_{M''_p} = U_i - c_p$ (see Fig. 1). M'_p is assigned to π_p and M''_p is assigned to π_{p+1}. For instance, in Fig. 1, $S_{p-1} = \{\tau_b, \tau_c, \tau_d\}$ and τ_e is split in M'_{p-1} and M''_{p-1}.

When the scheduler determines the execution time allocated to each task, the tasks in S_j are treated as a single task. Hence, we name S_j the *supertask* of processor π_j and the tasks in S_j are called *component tasks* of S_j. The utilization of S_j is denoted $U_{S_j} \stackrel{\text{def}}{=} \sum_{\tau_i \in S_j} U_i$.

After the assignment, EKG schedules the migratory tasks and supertasks using a DP-Fair technique [4]. The time is divided in time slices bounded by two successive task deadlines. At any time t, let TS^k denote the k^{th} time slice after time t (Note that, TS^k refers to different time slices as the time progresses. However, for the sake of readability, we write TS^k instead of TS$^k(t)$). The length of TS^k is $L^k \stackrel{\text{def}}{=} d_k(t) - d_{k-1}(t)$, where $d_k(t)$ is τ_k's deadline. In each time slice of length L^k, every supertask S_j and migratory task M'_j or M''_j, executes for $\ell^k_{S_j} = U_{S_j} \times L^k$, $\ell^k_{M'_j} = U_{M'_j} \times L^k$ and $\ell^k_{M''_j} = U_{M''_j} \times L^k$ time units respectively (see Fig. 1(b)). We say that $\ell^k_{S_j}$ is the *local execution time* of supertask S_j in time slice TS^k (and similarly for M'_j and M''_j).

Whenever, a supertask S_j is scheduled on processor π_j, the EDF algorithm is used to decide which of its component tasks will actually be executed on π_j. EKG can therefore be seen as a two-levels scheduling algorithm. On one hand, it uses a DP-Fair approach to schedule the supertasks and, on the other hand, it uses EDF to decide which component tasks to execute on the processor.

IV. Swapping Execution Time

As shown on Fig. 1(b), in every time slice, we have up to two preemptions on each processor π_j, caused by the two migratory tasks M''_{j-1} and M'_j, and one migration between processors π_j and π_{j+1} per migratory task M_j. On the other hand, since EDF is used to schedule the component tasks of any supertask S_j, the tasks included in S_j do not cause any preemption between two successive job arrivals (which correspond to the time slices boundaries). Hence, most of the preemptions and all migrations are caused by migratory tasks. It would therefore be valuable to remove the execution of these tasks in as many time slices as possible to reduce the run-time overheads. This is the goal of our swapping algorithm. Fig. 1(c) shows how the schedule might change after swapping and that preemptions and migrations may be reduced. Note that we will perform a swap only if it does not impact the correctness of the schedule previously built by EKG. Hence, *our algorithm does not affect the optimality of EKG.*

The swapping algorithm is applied at each time t corresponding to a task deadline (i.e., at the beginning of each time slice). We consider the current time slice TS^1 as the time interval extending from t to $d_1(t)$ (recall that the tasks are ordered in an increasing deadline order).

The algorithm follows four different rules:

***Rule** 1:* Maximize the execution of the migratory task M''_{j-1} on processor π_j in the current time slice TS^1.

***Rule** 2:* Minimize the execution of the migratory task M'_j on processor π_j in the current time slice TS^1.

***Rule** 3:* Avoid any intra-job parallelism.

***Rule** 4:* Ensure that every task will be able to complete its execution before its deadline.

The main idea of Rule 1 is to complete the execution of M''_{j-1} as early as possible. On the other hand, Rule 2 tries to delay as much as possible the execution of M'_j. The goal is that M''_{j-1} appears only in the earlier time slices after each of its new job arrival, and that M'_j appears only in the later time slices just prior to each job's deadline. Ideally, execution can be eliminated altogether in some time slices (see Fig. 1(c)), thereby eliminating the associated preemptions and migrations. Furthermore, reducing M'_j's execution in the current time slice increases the interval in which M''_{j-1} can execute in TS^1 giving the opportunity to M''_{j-1} to complete earlier.

Assuming that a task set τ is schedulable under EKG, Rules 3 and 4 ensure that the schedule remains correct after executing the swapping algorithm (i.e., we do not perform a swap if it could impact the validity of τ's schedule).

A. Principles Ensuring Correctness

Before we formally describe our swapping algorithm in the next section, we first present some principles we must follow in order to ensure that we never violate Rule 4.

***Principle** 1:* For any migratory task M_j, we cannot swap execution time between TS^1 and any time slice *subsequent* to the deadline of the current active job of the migratory task (i.e., a time slice TS^k such that $d_k(t) > d_{M_j}(t)$). Indeed, the time reserved for M_j after $d_{M_j}(t)$ is dedicated to the execution of "future" jobs of M_j that are not yet active in TS^1.

***Principle** 2:* If we increase (respectively decrease) the local execution time ℓ^1_i of a task τ_i by Δ_i time units in the current time slice TS^1, we have to decrease (respectively increase)

the local execution time ℓ_i^k by the same quantity Δ_i in a time slice TS^k such that $k > 1$ and TS^k is before $d_i(t)$ (according to Principle 1). That is, the time reserved for the execution of a task must remain constant for the whole schedule.

Principles 1 and 2 give us a straight forward approach for enforcing all migratory tasks' adherence to Rule 4. The approach for supertasks requires more thought. Indeed, each component task τ_i of a supertask S_j has its own deadline $d_i(t)$ and its own remaining execution time $r_i(t)$. Hence, we must ensure that S_j has enough time reserved on π_j to fulfill the execution of *all* component tasks before their respective deadlines. Moreover, if $d_i(t)$ is the deadline of the current job of a component task τ_i of S_j, then a part of the time reserved for S_j in time slices *subsequent* to $d_i(t)$ is dedicated to the execution of "future" jobs of τ_i. These jobs are not yet active within the interval $[t, d_i(t))$. Hence, we must be sure that we keep enough time after $d_i(t)$ to execute these future jobs.

Let F_j^k be the tasks in S_j with a deadline before $d_k(t)$ (i.e., $F_j^k = \{\tau_i \in S_j \mid d_i(t) < d_k(t)\}$). The time reserved in TS^k for any task τ_i in F_j^k, is dedicated to future jobs of τ_i. Hence, the local execution time $\ell_{S_j}^k$ allocated to supertask S_j within time slice TS^k can be separated into two parts —namely, the time reserved for the future jobs (i.e., jobs of tasks in F_j^k), denoted $f_{S_j}^k$, and the time allocated to active jobs at time t (i.e., jobs of tasks in $S_j - F_j^k$), denoted $a_{S_j}^k$. That is, we have $\ell_{S_j}^k = a_{S_j}^k + f_{S_j}^k$. Since, by definition, future jobs have not yet arrived at time t, we cannot swap execution time with these jobs. Therefore, the value of $f_{S_j}^k$ can never change. When using a DP-Fair approach such as EKG, $f_{S_j}^k$ is equal to the utilizations of the tasks in F_j^k multiplied by the length of the time slice (i.e., $f_{S_j}^k \stackrel{\text{def}}{=} L^k \times \sum_{\tau_i \in F_j^k} U_i$) [4], [5]. Therefore, we get the following two principles:

Principle 3: In any time slice TS^k, we cannot decrease the execution time of a supertask S_j by more than $a_{S_j}^k = \ell_{S_j}^k - f_{S_j}^k$.

Principle 4: Assuming that $d_q(t)$ is a job deadline, then, to respect Rule 4, we must ensure that the remaining execution of the currently active jobs never exceeds the time reserved for those jobs. Specifically, for each $q \leq n$,

$$\sum_{k=1}^{q} a_{S_j}^k \geq \sum_{\substack{\tau_i \in S_j \\ d_i(t) \leq d_q(t)}} r_i(t).$$

Note that after a swap between TS^1 and TS^k, Principle 4 requires the local execution time $\ell_{S_j}^1$ of S_j in TS^1 to be large enough to ensure , for each TS^k, that there is enough time allocated during TS^1 to TS^k to meet the demand. That is,

$$\ell_{S_j}^1 \geq \bar{a}_{S_j}^1 = \max_{1 \leq q < k} \Big(\sum_{\substack{\tau_i \in S_j \\ d_i(t) \leq d_q(t)}} r_i(t) - \sum_{v=2}^{q} a_{S_j}^v \Big)$$

B. Swapping Algorithm Description

Initially, EKG is used to assign the tasks among the processors and compute the local execution time of every migratory task and supertask in the n time slices bounded by the n current task deadlines at time t. Then, at time $t = 0$ and at any time t corresponding to a job deadline (i.e., a time slice boundary), our swapping algorithm is executed as shown in Algorithm 1[2]. This algorithm browses the time slices TS^2 to TS^n and updates the local execution times by performing swaps (if possible) with the current time slice TS^1 in accordance to Rules 1 through 4. We compute the swapped quantities starting with processor π_m and ending with processor π_1. On each processor π_j, we first perform the swap for the migratory tasks M'_j and M''_{j-1} (ignoring parallelism in TS^1 for the moment) (lines 2 to 5 in Algorithm 1). However, because Rule 3 is ignored during the first pass, some intra-job parallelism could arise. Therefore, we correct the values of the local execution times on each processor to remove the parallelism and respect Rule 3 (lines 7 to 9). Finally, we update the local execution time of the supertask S_j (line 10).

Algorithm 1: Swapping algorithm.

```
1  for k := 2 to n do
2      for j := m to 1 do
3          Swap between ℓ^1_{M'_j} and ℓ^k_{M'_j} using Lemma 1;
4          Swap between ℓ^1_{M''_{j-1}} and ℓ^k_{M''_{j-1}} using Lemma 2;
5      end
6      for j := 1 to m do
7          if There is parallelism between ℓ^1_{M'_{j-1}} and ℓ^1_{M''_{j-1}} then
8              Correct values using Eq. 1;
9          end
10         Update a^1_{S_j} and a^k_{S_j} using Eq. 2 ;
11     end
12 end
```

We will now derive the maximum time that can be swapped between TS^1 and TS^k for migratory tasks on processor π_j. According to Rules 1 and 2, we will increase the local execution time $\ell_{M''_{j-1}}^1$ of M''_{j-1} by $\Delta_{M''_{j-1}}^k$ in TS^1 and decrease $\ell_{M'_j}^1$ by $\Delta_{M'_j}^k$. Moreover, as stated in Principle 1, in order to respect Rule 4, we must correspondingly decrease $\ell_{M''_{j-1}}^k$ by $\Delta_{M''_{j-1}}^k$ and increase $\ell_{M'_j}^k$ by $\Delta_{M'_j}^k$ in TS^k.

Lemma 1: The maximum execution time of the migratory task M'_j that can be swapped from time slice TS^1 to time slice TS^k is

$$\Delta_{M'_j}^k = \min \left\{ \ell_{M'_j}^1 \ , \ \left(\ell_{M''_{j-1}}^k + a_{S_j}^k \right) \ , \ \left(L^k - \ell_{M'_j}^k - \ell_{M''_j}^k \right) \right\}$$

if $d_{M_j}(t) \geq d_k(t)$. Otherwise, $\Delta_{M'_j}^k = 0$.

Proof: We cannot swap more from TS^1 than what is allocated to M'_j. Hence, $\Delta_{M'_j}^k \leq \ell_{M'_j}^1$. Similarly, increasing M'_j's allocation in TS^k will decrease the time allocated to M''_{j-1} and S_j. Therefore, we cannot swap more into TS^k than what is allocated to M''_{j-1} and S_j. That is, $\Delta_{M'_j}^k \leq \ell_{M''_{j-1}}^k + a_{S_j}^k$ (see Fig. 2 (a)). Furthermore, from Rule 3, we

[2]Notice that at such an instant t, at least one new job arrives. Therefore, according to the deadlines of these jobs, one new time slice is created by newly released job. Hence, the local execution time for all tasks must be initialized in these new time slices.

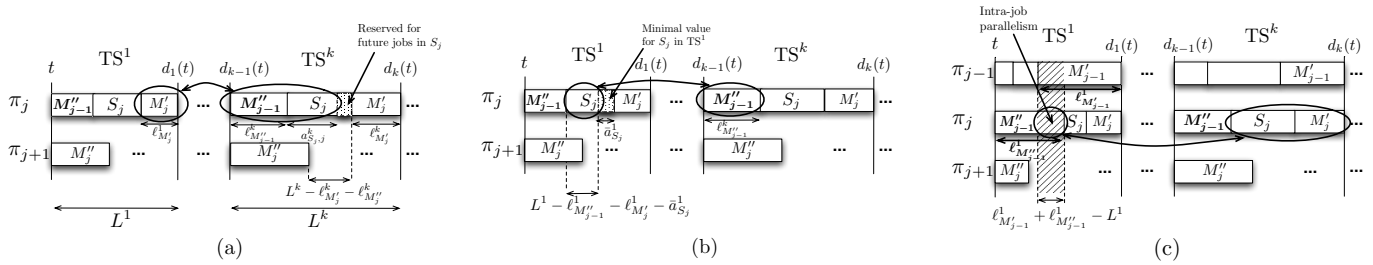

Fig. 2. Task swapping according to Algorithm 1.

forbid any intra-job parallelism. Therefore, because $\ell^k_{M''_j}$ has already been computed and will remain fixed, we must have $\Delta^k_{M'_j} \leq L^k - \ell^k_{M'_j} - \ell^k_{M''_j}$.

Finally, we stated in Principle 1 that we cannot swap any time from M_j between TS^1 and a time slice subsequent to $d_{M_j}(t)$. That is, $\Delta^k_{M'_j} = 0$ for such time slices. ■

Lemma 2: The maximum execution time of the migratory task M''_{j-1} that can be swapped from time slices TS^k to time slice TS^1 is

$$\Delta^k_{M''_{j-1}} = \min\left\{\ell^k_{M''_{j-1}}\ ,\ \left(L^1 - \ell^1_{M''_{j-1}} - \ell^1_{M'_j} - \bar{a}^1_{S_j}\right)\right\}$$

if $d_{M_{j-1}}(t) \geq d_k(t)$. Otherwise, $\Delta^k_{M''_{j-1}} = 0$.

Proof: We cannot swap more from TS^k than what is allocated to M''_{j-1}. Therefore, $\Delta^k_{M''_{j-1}} \leq \ell^k_{M''_{j-1}}$. Similarly, we cannot swap more from TS^1 than what is available. Since M''_{j-1} , M'_j and S_j must execute $\ell^1_{M''_{j-1}}$, $\ell^1_{M'_j}$ and $\bar{a}^1_{S_j}$ time units, that leaves $\Delta^k_{M''_{j-1}} \leq \left(L^1 - \ell^1_{M''_{j-1}} - \ell^1_{M'_j,} - \bar{a}^1_{S_j}\right)$.

Finally, we stated in Principle 1 that we cannot swap any time from M_{j-1} between TS^1 and a time slice subsequent to $d_{M_{j-1}}(t)$. That is, $\Delta^k_{M''_{j-1}} = 0$ for such time slices. ■

In case of intra-job parallelism between M'_{j-1} and M''_{j-1}, we can correct the values of the local execution times by adjusting the $\Delta^k_{M''_{j-1}}$ and $\Delta^k_{M'_j}$ quantities (see Fig. 2 (c)). In this situation, we decrease $\Delta^k_{M''_{j-1}}$ so that task M'_{j-1} starts to execute at the exact moment when M''_{j-1} finishes on processor π_j (i.e., M_{j-1} executes for exactly L^1 times units during TS^1). This frees up some time which can be spent either executing S_j or M'_j. By Rule 2, our preference would be to avoid increasing M'_j. However, Principle 3 states that we cannot decrease the execution of S_j in TS^k for more than $a^k_{S_j}$ time units. Therefore, if $a^k_{S_j}$ is too small then we may need to increase M'_j in TS^1. This gives the following adjustments to $\Delta^k_{M''_{j-1}}$ and $\Delta^k_{M'_j}$.

$$\begin{cases} \Delta^k_{M''_{j-1}} \leftarrow \Delta^k_{M''_{j-1}} - \left(\ell^1_{M'_{j-1}} + \ell^1_{M''_{j-1}} - L^1\right) \\ \Delta^k_{M'_j} \leftarrow \Delta^k_{M'_j} - \max\left\{0\ ,\ \ell^1_{M'_{j-1}} + \ell^1_{M''_{j-1}} - L^1 - a^k_{S_j}\right\} \end{cases} \quad (1)$$

Once we have computed the new values of the local execution times of task M''_{j-1} and M'_j on processor π_j, we must update $a^1_{S_j}$ and $a^k_{S_j}$ so that the total execution time remains constant in each time slice. We therefore get

$$\begin{cases} a^1_{S_j} \leftarrow a^1_{S_j} + \Delta_{M'_j} - \Delta_{M''_{j-1}} \\ a^k_{S_j} \leftarrow a^k_{S_j} - \Delta_{M'_j} + \Delta_{M''_{j-1}} \end{cases} \quad (2)$$

V. Conclusion and Future Works

In this paper we have presented a swapping algorithm which modifies the local execution time that the EKG algorithm allocates to the tasks in each time slice. Hence, we can reduce the run-time overheads incurred by this algorithm and improve its practicality for real applications.

Many improvements can be considered for this first swapping algorithm. We believe that, by slightly, modifying the algorithm, we could greatly reduce the number of time slices in the schedule. That is, we can drastically improve the number of preemptions, migrations and scheduling points. Moreover, the presented algorithm always use the same heuristic in each time slice to modify execution times allocated to tasks. We would like to investigate the impact of varying the used heuristics as the schedule progresses. Finally, we plan to extend our swapping algorithm for sporadic tasks (using the sporadic version of EKG).

References

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

[2] J. H. Anderson and A. Srinivasan, "Early-release fair scheduling," in *ECRTS 2000*, 2000, pp. 35–43.

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

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

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

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

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

[8] B. B. Brandenburg and J. H. Anderson, "On the implementation of global real-time schedulers," in *RTSS '09*, 2009, pp. 214–224.

[9] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers," in *RTSS '10*, 2010, pp. 14–24.

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