CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

Multiprocessor real-time scheduling with a few migrating tasks

J. Augusto Santos-Jr.
George Lima
Konstantinos Bletsas
Shinpei Kato

CISTER-TR-131204
Version:
Date: 12-05-2013

Multiprocessor real-time scheduling with a few migrating tasks

J. Augusto Santos-Jr., George Lima, Konstantinos Bletsas, Shinpei Kato

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.cister.isep.ipp.pt

Abstract

We present HIME, a new EDF-based semi-partitioned scheduling algorithm which allows at most one migrating task per processor. In a system with m processors, this arrangement limits the migrating tasks to at most m/2 and the number of migrations per job to at most m-1. HIME has a utilisation bound of at least 74.9%, and can be configured to achieve 75%, the theoretical limit for semi-partitioned schemes with at most m/2 migrating tasks. Experiments show that the average system utilisation achieved by HIME is about 95%.

Multiprocessor real-time scheduling with a few migrating tasks

J. Augusto Santos-Jr.*, George Lima*, Konstantinos Bletsas† and Shinpei Kato‡
*Computer Science Department, Federal University of Bahia, Brazil
†CISTER/INESC-TEC, ISEP, Porto, Portugal
‡Information Engineering Department, Nagoya University, Japan

***Abstract*—We present HIME, a new EDF-based semi-partitioned scheduling algorithm which allows at most one migrating task per processor. In a system with m processors, this arrangement limits the migrating tasks to at most $m/2$ and the number of migrations per job to at most $m-1$. HIME has a utilisation bound of at least 74.9%, and can be configured to achieve 75%, the theoretical limit for semi-partitioned schemes with at most $m/2$ migrating tasks. Experiments show that the average system utilisation achieved by HIME is about 95%.**

1. INTRODUCTION

Algorithms for scheduling n sporadic tasks on m identical processors to meet deadlines were traditionally classified as *partitioning* or *global scheduling* [11]. Partitioning divides the task set into disjoint subsets; each subset is assigned to a respective processor and scheduled by some uniprocessor algorithm such as Earliest-Deadline-First (EDF) and Fixed Priority Scheduling (FPS). Global scheduling maintains a single run queue for all tasks and, at any given instant, the m highest priority tasks execute, each on one of the m processors. Hence, under global scheduling, tasks may migrate, even halfway through the execution. Partitioning offers simplicity and reuse of techniques from uniprocessor scheduling but disallowing migrations inherently limits the utilisation bound to 50% at most [11]. Conversely, some global scheduling algorithms achieve a utilisation bound of 100% at the expense of higher scheduling overheads and/or scalability issues.

The novel *semi-partitioning* paradigm aims to combine the best aspects of partitioning (efficient implementation; low dispatching overheads) with those of global scheduling (efficient use of available processing capacity). Various semi-partitioned scheduling schemes have been devised. A few are based on fixed-priority scheduling but most are based on EDF, some of which achieve high schedulability [4], [5], [9]. However, what is left open for the semi-partitioning paradigm is a *utilisation bound* problem. It is a trade-off that algorithms designed to meet a high utilisation bound pay the cost of migration overheads [5] while simplified algorithms compromise the worst-case schedulability, if not the average schedulability [4], [9]. In this paper we follow this trend, describing a highly efficient semi-partitioned EDF-based approach reducing the number of migrating tasks as compared to the start-of-art.

Contribution: In this work, we formulate a new semi-partitioned scheduling algorithm called HIME, which stands for HIghest-priority Migration managed by EDF. HIME allows at most one migrating task per processor and employs very simple dispatching: a migrating task always executes at the highest priority over all other tasks, which are scheduled by EDF. This can be achieved using a standard uniprocessor EDF scheduler, via manipulation of the deadline of the migrating task on each processor (*i.e.*, reporting a shorter deadline to the scheduler, as in [5]). Allowing at most one migrating task per processor means that there may exist no more than $m/2$ migrating tasks. Additionally, as will become evident, no job may migrate more than $m-1$ times. Such a design makes good sense, since having frequent migrations and/or too many migrating tasks may in practice lead to degraded performance, due to increased contention on processor caches.

For HIME, we prove a least utilisation bound of 74.9%, which can be raised to 75% with a simple optimisation; this matches, as we also prove, the theoretical limit for the class of algorithms with at most $\lfloor m/2 \rfloor$ migrating tasks.

Organisation: Section 2 first presents the underlying principles of HIME and next the algorithm itself. Section 3 contains the theoretical derivation of the least utilisation bound of HIME. Section 4 describes possible improvements. Section 5 experimentally assesses the scheduling potential of HIME *vs.* that of other schemes. Our conclusions are drawn in Section 6.

2. DESCRIPTION OF HIME

A. Task model

We consider a set Γ of n independent sporadic tasks (τ_1 to τ_n) scheduled on m identical processors (P_1 to P_m). A task τ_i is denoted by a tuple (C_i, T_i), where $C_i \leq T_i$ represents its worst-case execution time and T_i is its minimum inter-arrival time, also called period. The utilisation of a task τ_i is denoted by $u_i \stackrel{\text{def}}{=} C_i/T_i$ while that of an overall task set $\Gamma' \subseteq \Gamma$ is denoted by $U(\Gamma') \stackrel{\text{def}}{=} \sum_{\tau_i \in \Gamma'} u_i$.

If a task τ_i arrives at time t, it must be granted C_i units of processor time within $[t, t+T_i)$. Namely, we assume an *implicit-deadline* task model. During their execution, tasks may be preempted and may migrate between processors but each individual task is not allowed to execute at the same time on distinct processors. The actual costs of preemptions and migrations are outside the scope of this paper.

B. Task splitting and migration

As aforementioned, under HIME, no processor is used by more than one migrating task. This arrangement breaks up

the system into multiple clusters (disjoint sets of processors), each of which may contain a single respective migrating task (Figure 1). Within each cluster, the migrating task always executes at the highest priority over other tasks and uses each of the processors for a specific fraction of its execution time, after which it simply migrates to the next processor.

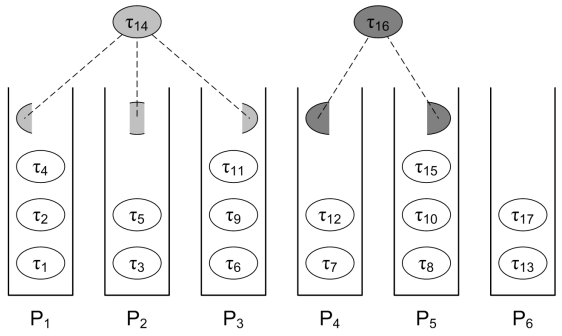


Fig. 1. An example of clustered task splitting

The amount of a respective fraction of its overall worst-case execution time that a migrating task may spend on each processor is calculated offline, during the task assignment and splitting stage. While it is desirable to maximise that fraction for an efficient processor utilisation, the interference by the migrating task should not compromise the schedulability of background (*i.e.*, non-migrative) tasks. Therefore, each "piece" of a migrating task is sized according to a uniprocessor schedulability test. Although this test is only sufficient (unlike the exact tests used by conceptually related algorithms EDF-WM [9] and C=D [5]), it is polynomial in time complexity and, as we will show, performs well. Additionally, our derivation of the utilisation bound of HIME depends on this particular test, presented next:

Theorem 1 (Basic sufficient schedulability test): Let τ_0 be a task scheduled at the highest priority on some processor P_p, together with a set Γ_p of implicit-deadline background tasks, scheduled under EDF. Additionally, assume that $T_0 \leq T_j,\ \forall \tau_j \in \Gamma_p$. Then, no deadlines can be missed as long as

$$u_0 \leq \frac{1 - U(\Gamma_p)}{1 + U(\Gamma_p)} \tag{1}$$

Proof: It is known from Theorem 3 in [12] and [13] (ported as Theorem 7 in the Appendix) that, for the case described above, no deadlines can be missed if

$$u_0 \leq \frac{1 - U(\Gamma_p)}{1 + \frac{U(\Gamma_p)}{\left\lfloor \frac{\min_{\tau_j \in \Gamma_p} T_j}{T_0} \right\rfloor}}$$

Since $T_0 \leq \min_{\tau_j \in \Gamma_p} T_j$ (from the assumption), the above sufficient condition for schedulability can be relaxed to (1). ■

When applying the above schedulability test in order to size a piece of a split task, the task τ_0 in Theorem 1 corresponds to the split task piece and Γ_p corresponds to the set of non-migrative tasks assigned to processor P_p. Then, for a task τ_i split over k processors (P_q to P_{q+k-1}), its piece on processor P_p is sized such that its utilisation u_i^p is

$$u_i^p = \begin{cases} \sigma(U(\Gamma_p)) \stackrel{\text{def}}{=} \frac{1-U(\Gamma_p)}{1+U(\Gamma_p)}, & \text{for the first } k-1 \text{ pieces;} \\ u_i - \sum_{p=q}^{q+k-2} u_i^p, & \text{for the last } (k^{th}) \text{ piece.} \end{cases} \tag{2}$$

Note however that Equation (2) is only safe if $T_i \leq T_j$, $\forall \tau_j \in \Gamma_p$, reflecting the assumptions of Theorem 1. The offline task assignment and splitting algorithm (examined in detail in Section 2-C), ensures this by design, by selecting as migrating task the task with the shortest interarrival time on each cluster.

Of use, is also the following, more conservative, formula for sizing u_i^p:

Lemma 1: It is safe to set u_i^p to $\alpha(U(\Gamma_p)) \stackrel{\text{def}}{=} 2(\sqrt{2}-1) - U(\Gamma_p)$ instead of $\sigma(U(\Gamma_p))$.

Proof: From Theorem 1, processor P_p is schedulable if $u_i^p \leq \sigma(U(\Gamma_p))$. In turn, $U(\Gamma_p) + \sigma(U(\Gamma_p)) = U(\Gamma_p) + \frac{1-U(\Gamma_p)}{1+U(\Gamma_p)} = \frac{U^2+1}{1+U(\Gamma_p)}$ and, using standard differential calculus (see Remark 5 in the Appendix), the minimum of this expression over $U(\Gamma_p) \in [0,1]$ is $2(\sqrt{2}-1)$. ■

Intuitively, Lemma 1 states that the least schedulable processor utilisation in the presence of multiple EDF-scheduled background tasks (under interference of a single higher-priority task) is lower-bounded by the respective least utilisation bound when there is just a single background task – or equivalently the least utilisation bound of $2(\sqrt{2}-1)$ for two tasks scheduled by rate-monotonic [10]. In that same seminal paper, Liu and Layland had explored this exact mixed scheduling model, and although they had stopped short of identifying the asymptotic bound, they made a similar conjecture[1].

C. Outline of task assignment and splitting

The high-level pseudocode in Figure 2 outlines the task assignment and splitting procedure. The tasks (indexed by non-increasing utilisation) are integrally assigned, one by one, using First-Fit bin-packing, until either all tasks are assigned or some task τ_i cannot be integrally assigned to any processor, with schedulability preserved subject to existing task assignments. At that point, the algorithm resorts to task splitting.

For the purposes of task splitting, a variable q holds the index of the first processor onwards from which the next cluster is to be formed. Prior to task splitting, processors P_q to P_m are re-indexed in terms of non-decreasing $U(\Gamma_p)$ (line 6). One *naive* way of proceeding, at this point, would be to attempt to split τ_i (the same task that could not be assigned integrally, earlier) in k pieces (as many as needed) over processors P_q, P_{q+1}, ..., P_{q+k-1} respectively, sizing each piece such that each processor remains schedulable. However, this approach includes a problem. Namely, the piece sizing formula (2) relies on the migrating task having a T no greater

[1]Quoting from [10]: "Although a closed form expression for the least upper bound to processor utilization has not been found for the mixed scheduling algorithm, this example strongly suggests that the bound is considerably less restrictive for the mixed scheduling algorithm than for the fixed priority rate-monotonic scheduling algorithm. The mixed scheduling algorithm may thus be appropriate for many applications."

```
1.  //tasks are indexed by non-increasing u_i
2.  int q=1; //stores index of processor to split onwards from
3.  for (each task τ_i)
4.  {try assigning τ_i integrally to some processor in {P_1,...,P_m} using First-Fit;
5.   if (τ_i could not be assigned to any processor with
         schedulability preserved, subject to existing assignments)

6.    {re-index processors P_q to P_m by non-decreasing utilisation;
7.     k'=select_processors_for_next_cluster(); //returns upper bound on cluster size
8.     j= index_of_task_with_shortest_T_in(Γ_q ∪ ... ∪ Γ_{q+k'-1});
9.     if (T_i > T_j)
10.    {undo_assignment_of(τ_j);
11.     assign τ_i to the processor P_r where τ_j was previously assigned;
12.     re-index processors P_q to P_{q+k'-1} by non-decreasing U(Γ_p); //if affected by swap
13.    }
14.    else //it is τ_i that will be split
15.     j=i;

16.    int k=split_task(τ_j); //from P_q onwards, using Eq. (2) to size pieces;
17.    if (k==0) //signifies that we ran out of processors!
18.     return(FAILURE);
19.    else
20.     q=q+k; //k is the number of processors over which τ_j was split
21.   }
22. }

    //this line is reached only if all tasks were assigned (integrally or split)
23. return(SUCCESS);
```

Fig. 2. Pseudocode for task assignment/splitting stage of the HIME scheduling algorithm

than that of any other task, on any of the processors that it uses. In other words, the migrating task must have the shortest T on its cluster – and this may not be the case for τ_i, since the task set is indexed (line 1) by non-increasing utilisation (u), not by non-increasing interarrival time (T). Although this could be resolved by indexing the tasks by non-increasing T instead, this would degrade the utilisation bound. Hence, we proceed differently:

Via a technique explained later (Section 2-D), we obtain (line 7) an upper bound k' on the number of processors needed for the next cluster *before* the identity of the split task is even decided. In the worst case, it could be all $m - q + 1$ candidate processors (P_q to P_m). This bound k' is calculated by function select_processors_for_next_cluster(), which, also reindexes the processors such that P_q to $P_{q+k'-1}$ are really the ones we want for the next cluster (in terms of what's best for the utilisation bound).

Now that the processors in the next cluster are known, we can check whether $T_i \leq T_j,\ \forall \tau_j \in \Gamma_q \cup \ldots \cup \Gamma_{q+k'-1}$. If this holds, then τ_i is chosen as the task to split; else, we do a *task swap* (lines 8-12):

Let τ_j be the task with the minimum interarrival time over $\Gamma_q \cup \ldots \cup \Gamma_{q+k'-1}$ and let P_r be the processor where it was assigned. As a next step, the assignment of τ_j is undone; τ_i is assigned to P_r in its place; and τ_j is selected as the task to split over the cluster under formation. Note that this swap is always possible (*i.e.*, safe for the schedulability of P_r) because $u_i \leq u_j$, from the task ordering); hence $U(\Gamma_r)$ cannot increase as a result. Next, the task selected (τ_j, without loss of generality) is split up in pieces, assigned to P_q onwards and sized according to Equation (2). This splitting is performed by function split_task(τ_j), invoked at line 16. The specific internal workings of this function are presented in pseudocode in Figure 4. Similarly, they perform some processor reindexing and a local optimisation for choosing the processor for the last piece of the split task.

Note that since k' is only an upper bound on the number (k) of processors of the cluster under formation, we may eventually need fewer processors than that. In any case, after τ_j is split, variable q is incremented by k (line 20).

After completing the assignment of a split task and the formation of the respective cluster, the algorithm continues with the assignment of remaining tasks, resorting to splitting, in the manner described, whenever needed. The algorithm can only fail if some task τ_j cannot be split over the remaining processors (line 18).

D. Properties of cluster formation

The correctness of the algorithm of Figure 2 and the utilisation bound proven in Section 3 rely on an upper bound k' for the number of processors in the cluster, computed *before* its formation. This bound k' is derived using the algorithm in Figure 3. To summarise, that function computes k' by "pretending" that (i) the task to be split is τ_i (*i.e.*, the same task which earlier could not be assigned integrally) and (ii) that its pieces on each processor can be sized using the function $\sigma(U(\Gamma_p))$ (as defined in Equation (2)), even if its T_i is not the smallest in the cluster under formation. For the last (k'^{th}) processor, the conservative sizing formula $\alpha(U(\Gamma_p))$ from Lemma 1 is used instead, for resilience in case of a task swap.

The fact that this derivation of k' is safe, is proven by Lemma 3, which relies on Lemma 2:

```
1.  int select_processors_for_next_cluster()
2.  {//P_q .. P_m already indexed by U(Γ_p) ↑
3.   int k'=1;
4.   int p=q; //the new cluster starts from P_q
5.   float U=u_i; //u of τ_i that could not be assigned integrally

     //the loop below calculates a provisional estimate for k'
     //and identifies the first k'-1 processors
6.   while ((U>σ(U(Γ_p))) AND (p ≤ m))
7.   {//consider P_p as member of the next cluster
8.    U=U-σ(U(Γ_p));
9.    k'=k'+1;
10.   p=p+1;
11.  }
12.  //provisional k' has been calculated; now, attempting
     //a local optimisation for the last (k'^th) processor

     //P_q..P_m are indexed by U(Γ_p) ↑, hence also by α(U(Γ_p)) ↓
     //The loop below identifies the P_p with the highest U(Γ_p),
     //among those with α(U(Γ_p)) >U (if any exist).

13.  for (p=m; p≥q+k'-1; p--) //traverse P_m → P_{q+k'-1}
14.  {if (α(U(Γ_p)) ≥U) //if the last piece fits, acc. to test of Lem. 1
15.    break;
16.  }
17.  if (p≥q+k'-1)
18.  {reindex P_p to become P_{q+k'-1};
      //maintaining relative order of other processors
19.    return k';
20.  }
21.  else //no such processor found
22.   return m-q+1;
23. }
```

Fig. 3. Pseudocode for selecting processors for the next cluster

Lemma 2: Whenever the algorithm of Figure 2 resorts to task-splitting, just before select_processors_for_next_cluster() is invoked, it holds that $U(\Gamma_p) > \frac{1}{2},\ \forall q \leq p \leq m$.

Proof: There exist two complementary cases: If (Case 1) $u_i > \frac{1}{2}$, since every $P_p \in \{P_q \dots P_m\}$ has at least one task and each task τ_j assigned so far has $u_j \geq u_i$, the claim holds. Alternatively, if (Case 2) $u_i \leq \frac{1}{2}$, the claim follows from the fact that τ_i could not be assigned integrally on any P_p. ■

Lemma 3: If, during the execution of function select_processor_for_next_cluster(), line 19 (Figure 3) is reached, then, the subsequent call to the function split_task() will succeed in splitting the migrating task (whether this is the task τ_i which could not be assigned integrally earlier or the task τ_j with which it is swapped at lines 10-12 of Figure 2), over (a subset of) the processors $\{P_q, \dots, P_{q+k'-1}\}$ selected by select_processor_for_next_cluster().

Proof: We identify two complementary cases, corresponding to the branches of the "if-else" statement of line 5 (Figure 2). If (Case 1) τ_i (the task which could not be assigned integrally) is the actual task to be split ("else" branch), the proof is trivial. Therefore, we focus on Case 2, in which τ_i is swapped with another task τ_j, and τ_j is split instead.

Let P_r be the processor (as reindexed by select_processors_for_next_cluster()) where τ_j was initially assigned. Moreover let $U^{\text{bef}}(\Gamma_p)$ denote the utilisation of Γ_p before the swap of τ_j and τ_i and $U^{\text{aft}}(\Gamma_p)$, respectively, afterwards. Then, $S^{\text{bef}} \stackrel{\text{def}}{=} 1 - (U^{\text{bef}}(\Gamma_p) + \sigma(U^{\text{bef}}(\Gamma_p)))$ and

```
1.  int split_task(τ_j)
2.  {reindex P_q..P_{q+k'-1} by U(Γ_p)↑; //hence also σ(U(Γ_p)) ↓
3.   C=C_j;
4.   int p=1;
5.   while (p ≤ k')
6.   {if (C/T_j ≤ σ(U(Γ_p)))
7.     break; //local optimisation will be used for the last piece
8.    else
9.    {assign piece τ_j^{q+p-1}
       with C_j^{q+p-1}=T_i*σ(U(Γ_p)) to P_{q+p-1};
10.    C=C-C_j^{q+p-1};
11.    p=p+1;
12.   }
13.  }

14.  if (p>k') //we ran out of processors!
15.   return 0;
16.  else //assign last piece to most utilised processor possible
17.  {int k=p; //k ≤ k'
18.   for (p=m; p≥ q+k'-1; p--) //traverse P_m → P_{q+k-1}
19.   {if (σ(U(Γ_p)) ≥C/T_j) //if the last piece fits, by test of Th. 1
20.     reindex P_p to become P_{q+k-1};
       //maintaining relative order of other processors
21.    assign piece τ_j^{q+k-1} with C_j^{q+k-1}=C to P_{q+k-1};
22.    return k; //size of cluster
23.   }
24.  }
25. }
```

Fig. 4. Pseudocode for task splitting

$S^{\text{aft}} \stackrel{\text{def}}{=} 1 - (U^{\text{aft}}(\Gamma_p) + \sigma(U^{\text{aft}}(\Gamma_p)))$ denote the fraction of the capacity of P_p which is unusable for accommodating additional task utilisation, respectively before/after the swap.

In order to show that the migrating task can be accommodated over P_q to $P_{q+k'-1}$ even in the case of a task swap, it suffices to show that

$$S_r^{\text{aft}} - S_r^{\text{bef}} \leq 3 - 2\sqrt{2} - S_{q+k-1}^{\text{bef}}, \tag{3}$$

wherein the right-hand side represents the slack on P_{q+k-1} for accommodating (additional) utilisation by the migrating task. This slack results from the capacity of P_{q+k-1} for accommodating migrating tasks being pessimistically calculated by select_processor_for_next_cluster() as $\alpha(U^{\text{bef}}(\Gamma_p)) = (2-2\sqrt{2}) - U(\Gamma_p)$, instead of $\sigma(U^{\text{bef}}(\Gamma_p))$.

Given that $S_r^{\text{aft}} - S_r^{\text{bef}} \leq \max_{p=q}^{q+k-1} S_p^{\text{aft}} - \min_{p=q}^{q+k-1} S_p^{\text{bef}}$ and $\max_{p=q}^{q+k-1} S_p^{\text{aft}} \leq 3 - 2\sqrt{2}$ (from the fact that $U^{\text{aft}}(\Gamma_j) + \sigma(U^{\text{aft}}(\Gamma_j)$ is lower-bounded by $2(\sqrt{2}-1)$), it follows that:

$$S_r^{\text{aft}} - S_r^{\text{bef}} \leq 3 - 2\sqrt{2} - \min_{p=q}^{q+k-1} S_p^{\text{bef}} \tag{4}$$

Given that $U^{\text{bef}}(\Gamma_p) > \frac{1}{2}$ (Lemma 2) and that S_p^{bef} is decreasing over $[\sqrt{2}-1, 1]$ (see Remark 5 in the Appendix), it follows that $\min_{p=q}^{q+k-1} S_p^{\text{bef}} \geq S_{q+k-1}^{\text{bef}}$, since the function select_processors_for_next_cluster() selects processors in non-decreasing order of $U^{\text{bef}}(\Gamma_p)$. Combining this fact with Equation (4) yields $S_r^{\text{aft}} - S_r^{\text{bef}} \leq 3 - 2\sqrt{2} - S_{q+k-1}^{\text{bef}}$. ■

Theorem 2: If the algorithm of Figure 2 declares success, no task misses its deadline, when scheduled under our model.

Proof: By construction, since each successful task assignment (split or integral) preserves the schedulability of previously assigned tasks; else failure is declared. ■

E. Ilustration

The allocation scheme of HIME is now illustrated.

Example 1: Let Γ be a task set to be scheduled on $m = 4$ processors composed of the following tasks: $\tau_1 = \tau_2 = (2.04, 3), \tau_3 = \tau_4 = (1.34, 2)$, and $\tau_5 = (1.32, 2)$.

HIME first assigns the first four tasks integrally to distinct processors and detects that τ_5 cannot be assigned to a single processor (Figure 2). The first part of Table I illustrates these allocation steps. Then, HIME searches for processors so that a migrating task can be accommodated. In this example, all four processors are selected (Figure 3). Note that by Equation (2) $U(\tau_5) = 0.66$ is greater than what is available on any group of three processors after the first four tasks are allocated. As the period of τ_5 is not greater than the period of any other already allocated tasks, it can be set as a migrating task.

TABLE I
TASK-TO-PROCESSOR ASSIGNMENT FOR EXAMPLE 1.

	allocation of $\tau_1, \tau_2, \tau_3, \tau_4$			
Processor	P_1	P_2	P_3	P_4
Allocation	$\{\tau_1\}$	$\{\tau_2\}$	$\{\tau_3\}$	$\{\tau_4\}$
$U(\Gamma_p)$	0.6800	0.6800	0.6700	0.6700
$\sigma(U(\Gamma_p))$	0.1904	0.1904	0.1976	0.1976
	allocation of τ_5			
Allocation	$\{\tau_1, \tau_5^1\}$	$\{\tau_2, \tau_5^2\}$	$\{\tau_3, \tau_5^3\}$	$\{\tau_4, \tau_5^4\}$
$U(\Gamma_p) + u_i^p$	0.7962	0.8705	0.8676	0.8676
$\sigma(U(\Gamma_p)) - u_i^p$	0.1162	0.0	0.0	0.0

As will be shown in the next sections, the least upper bound on system utilisation provided by HIME is not higher than 75%. However, as seen in Example 1, which requires 85% of four processors, the average case is typically much higher.

3. UTILISATION BOUND

In this section, we prove the least utilisation bound of 74.9% for for HIME. First, we need some intermediate results:

Remark 1: If a task τ_i cannot be assigned integrally subject to existing assignments, then

$$u_i + U(\Gamma_p) > 1,\ p = q, \ldots, m \tag{5}$$

$$U(\Gamma_i^p) + U(\Gamma_i^{p'}) > 1,\ p, p' = q, \ldots, m, p \neq p' \tag{6}$$

Proof: From the bin-packing used. ∎

Remark 2: If a task τ_j is split over $k > 1$ processors (P_q to P_{q+k-1}) by the algorithm of Figure 2 under our system model, then the following relations hold:

$$u_j + U(\Gamma_p) > 1,\ p = q, \ldots, q + k - 1 \tag{7}$$

$$U(\Gamma_j^p) + U(\Gamma_j^{p'}) > 1\ p, p' = q, \ldots, q + k - 1, p \neq p' \tag{8}$$

$$\sum_{p=q}^{q+k-2} \frac{1 - U(\Gamma_p)}{1 + U(\Gamma_p)} < u_j \tag{9}$$

Proof: If τ_j was the task which earlier could not be assigned integrally, (7) and (8) hold due to Remark 1. Else, if τ_j was swapped with that task τ_i, (5) and (6) held for τ_i before the task swap. The swap does have the effect of modifying the $U(\Gamma_r)$ of the processor P_r where τ_j was previously assigned; however the fact that $u_j \geq u_i$ (from the task ordering) ensures that (7) and (8) hold. Further, (9) follows from the fact that τ_j requires k processors; $k - 1$ do not suffice. ∎

Remark 3: Let $x_1, x_2, \ldots, x_k$ be values such that $0 \leq x_j \leq 1$, $j = 1, 2, \ldots, k$. If $\bar{x} = \frac{1}{k}\sum_{j=1}^{k} x_j$, then

$$k\frac{1 - \bar{x}}{1 + \bar{x}} \leq \sum_{j=1}^{k} \frac{1 - x_j}{1 + x_j} \tag{10}$$

Proof: Let $f(x) = \frac{1-x}{1+x}, 0 \leq x \leq 1$. As $f(x)$ is convex in interval $[0, 1]$, Jensen's inequality [8] can be applied,

$$f(\bar{x}) = f\left(\frac{1}{k}\sum_{i=1}^{k} x_j\right) \leq \frac{1}{k}\sum_{i=1}^{k} f\left(x_j\right)$$

∎

Now, we start the derivation of the utilisation bound of HIME. The next few lemmas explore particular cases; they are combined by Theorem 3.

Lemma 4: If a migrating task τ_j is succesfully split over $k = 2$ processors (P_q and P_{q+1}), under our system model and the algorithm described earlier, then $u_i + U^{\text{bef}}(\Gamma_q) + U^{\text{bef}}(\Gamma_{q+1}) = u_j + U^{\text{aft}}(\Gamma_q) + U^{\text{aft}}(\Gamma_{q+1}) > \frac{3}{2}$, where τ_i is the task which earlier could not be assigned integrally (*i.e.*, possibly, $i \neq j$), and $U^{\text{bef}}(\Gamma_p)$ and $U^{\text{aft}}(\Gamma_p)$, respectively, denote the value of $U(\Gamma_p)$ before/after the task swap (if one occured; else they coincide).

Proof: By design, $u_i + U^{\text{bef}}(\Gamma_q) + U^{\text{bef}}(\Gamma_{q+1}) = u_j + U^{\text{aft}}(\Gamma_q) + U^{\text{aft}}(\Gamma_{q+1})$. The claim then follows from (7), (8). ∎

Remark 4: Let τ_i be a task that could not be assigned integrally at line 5 (Figure 2). Then, if function select_processors_for_next_cluster (Figure 3) subsequently returns k', it holds that $u_i > \sum_{p=q}^{q+k'-2} \sigma(U^{\text{bef}}(\Gamma_p))$

Proof: Given that select_processor_for_next_cluster() "simulates" the splitting of τ_i over processors with $U^{\text{bef}}(\Gamma_p)$ (and function $\sigma(U(\Gamma_p))$ is used to size the first $k' - 1$ pieces), (9) of Remark 2 (which is formulated for τ_j and $U^{\text{aft}}(\Gamma_p)$) also holds for τ_i and $U^{\text{bef}}(\Gamma_p)$, namely: $u_i > \sum_{p=q}^{q+k'-2} \sigma(U^{\text{bef}}(\Gamma_p))$. ∎

Lemma 5: If, during the execution of function select_processor_for_next_cluster(), line 19 (Figure 3) is reached, then

$$u_i + \sum_{p=q}^{q+k'-1} U^{\text{bef}}(\Gamma_p) > (k' - 1)\frac{1 + \bar{u}^2}{1 + \bar{u}} + U^{\text{bef}}(\Gamma_{q+k'-1}) \tag{11}$$

where τ_i is the task that could not be integrallly assigned earlier and $\bar{u} \stackrel{\text{def}}{=} \frac{1}{k'-1}\sum_{p=q}^{q+k'-2} U^{\text{bef}}(\Gamma_p)$.

Proof: From Remark 4: $\sum_{p=q}^{q+k'-2} \frac{1 - U^{\text{bef}}(\Gamma_p)}{1 + U^{\text{bef}}(\Gamma_p)} < u_i$.

Applying Remark 3, we obtain:

$$u_i > \sum_{p=q}^{q+k'-2} \frac{1 - U^{\text{bef}}(\Gamma_p)}{1 + U^{\text{bef}}(\Gamma_p)} \geq (k' - 1)\frac{1 - \bar{u}}{1 + \bar{u}}$$

Therefore, $u_i + \sum_{p=q}^{q+k'-1} U^{\text{bef}}(\Gamma_p) \geq$

$$U^{\text{bef}}(\Gamma_{q+k'-1}) + \sum_{p=q}^{q+k'-2} U^{\text{bef}}(\Gamma_p) + (k'-1)\frac{1-\bar{u}}{1+\bar{u}}$$
$$= U^{\text{bef}}(\Gamma_{q+k'-1}) + (k'-1)\bar{u} + (k'-1)\frac{1-\bar{u}}{1+\bar{u}}$$
$$= (k'-1)\frac{1+\bar{u}^2}{1+\bar{u}} + U^{\text{bef}}(\Gamma_{q+k'-1})$$

■

Lemma 6: If function select_processors_for_next_cluster() selects $k' > 3$ processors (P_q to $P_{q+k'-1}$) and function split_task(τ_j) indeed uses all of those (*i.e.*, $k = k'$), to split the migrating task τ_j in consideration , then

$$\frac{1}{k-1}\sum_{p=q}^{q+k-2} U^{\text{bef}}(\Gamma_p) > \frac{\sqrt{17}-3}{2} \tag{12}$$

Proof: Let $\bar{u} \stackrel{\text{def}}{=} \frac{1}{k-1}\sum_{p=q}^{q+k-2} U^{\text{bef}}(\Gamma_p)$ and τ_i be the task that could not be assigned integrally at line 5 (Figure 2). From Remark 4:

$$u_i > \sum_{p=q}^{q+k-2} \sigma(U^{\text{bef}}(\Gamma_p)) = \sum_{p=q}^{q+k-2} \frac{1-U^{\text{bef}}(\Gamma_p)}{1-U^{\text{bef}}(\Gamma_p)}$$
$$\stackrel{Rem.\ 3}{\Longrightarrow} u_i > (k-1)\frac{1-\bar{u}}{1+\bar{u}} \tag{13}$$

Since tasks are assigned by non-increasing utilisation: $u_i \leq U^{\text{bef}}(\Gamma_p)$, $\forall p \Rightarrow u_i \leq \bar{u}$. Combining this with (13),

$$\bar{u} > (k-1)\frac{1-\bar{u}}{1+\bar{u}} \Rightarrow \bar{u}^2 + k\bar{u} - (k-1) > 0$$
$$\Rightarrow \bar{u} > \frac{\sqrt{k^2-4(k-1)}-k}{2} \tag{14}$$

As the RHS of (14) is an increasing function of k, its minimum occurs when $k = 3$. Therefore, $\bar{u} > \frac{\sqrt{17}-3}{2}$ ■

Lemma 7: If function select_processors_for_next_cluster() selects $k' > 3$ processors and function split_task(τ_j) uses just $k < k'$ of those (P_q to P_{q+k-1}), to split the migrating task τ_j in consideration, then

$$\frac{u_j + \sum_{p=q}^{q+k-2} U^{\text{aft}}(\Gamma_p)}{k} > \frac{5}{6}$$

Proof: Let τ_i be the task which could not be assigned integrally, just prior to the invocation of select_processors_for_next_cluster(). Reasoning similarly as in the start of the proof of Lemma 6:

$$u_i > (k'-1)\frac{1-\bar{u}}{1+\bar{u}} \tag{15}$$

where $\bar{u} \stackrel{\text{def}}{=} \frac{1}{k-1}\sum_{p=q}^{q+k-2} U^{\text{bef}}(\Gamma_p)$.

Combining the fact that $u_i + \sum_{p=q}^{q+k-1} U^{\text{bef}}(\Gamma_p) = u_j + \sum_{p=q}^{q+k-1} U^{\text{aft}}(\Gamma_p)$, with (15), we obtain

$$u_j + \sum_{p=q}^{q+k-1} U^{\text{aft}}(\Gamma_p) > (k'-1)\frac{1-\bar{u}}{1+\bar{u}} + k\bar{\nu} \tag{16}$$

where $\bar{\nu} \stackrel{\text{def}}{=} \frac{1}{k}\sum_{p=q}^{q+k-1} U^{\text{bef}}(\Gamma_p)$.

But from (6) (Remark 1) it follows that $\bar{u}, \bar{\nu} > \frac{1}{2}$. Therefore, via substitution to (16), we obtain:

$$u_j + \sum_{p=q}^{q+k-1} U^{\text{aft}}(\Gamma_p) > \frac{k}{2} + \frac{k'-1}{3}$$
$$\Rightarrow \frac{u_j + \sum_{p=q}^{q+k-1} U^{\text{aft}}(\Gamma_p)}{k} > \frac{1}{2} + \frac{k'-1}{3k} \tag{17}$$

Since $k \not> k'$, the claim holds. ■

Lemma 8: If function select_processors_for_next_cluster() selects $k' > 3$ processors and function split_task(τ_j) uses all $k = k'$ of those to split the migrating task τ_j, then

$$\frac{u_j + \sum_{p=q}^{q+k-1} U^{\text{aft}}(\Gamma_p)}{k} > 2\left(\frac{\sqrt{17}}{3} - 1\right) \tag{18}$$

Proof: In this scenario, function split_task() works with the same set of processors selected by function select_processors_for_next__cluster(). Thus,

$$\frac{u_j + \sum_{p=q}^{q+k-1} U^{\text{aft}}(\Gamma_p)}{k} = \frac{u_i + \sum_{p=q}^{q+k-1} U^{\text{bef}}(\Gamma_p)}{k}$$

where τ_i is the task that could not be integrally assigned to a processor (line 5, Figure 2). Combining this with Lemma 5:

$$\frac{u_j + \sum_{p=q}^{q+k-1} U^{\text{aft}}(\Gamma_p)}{k} > \frac{k-1}{k}\frac{1+\bar{u}^2}{1+\bar{u}} + \frac{U^{\text{bef}}(\Gamma_{q+k-1})}{k}$$

We know from the search order (line 2 in Figure 3) used in function select_processors_for_next_cluster() that $U^{\text{bef}}(\Gamma_{q+k-1}) \geq \bar{u}$ and from Lemma 6 we have that $\bar{u} > \frac{\sqrt{17}-3}{2}$. Substituting these values in the RHS of the above inequality leads to (18). ■

Lemma 9: Let k' be the number of processors returned by select_processor_for_next_cluster() (Figure 2, line 7), when some task τ_i could not be integrally assigned in line 4. Then, if HIME subsequently fails (line 18) upon attempting to split a task τ_j, it holds that $u_j + \sum_{p=q}^{q+k'-1} U^{\text{aft}}(\Gamma_p) > 2(\sqrt{2}-1)k'$.

Proof: This proof need not explore separate cases, whether τ_j is the same task as τ_i (in which case $U^{\text{bef}}(\Gamma_p) = U^{\text{aft}}(\Gamma_p)$ for all P_p considered) or there is a task swap.

HIME returns failure when function split_task(τ_j) returns $k = 0$ because k' processors were not sufficient to assign τ_j as a migrating task. This means that

$$u_j > \sum_{p=q}^{q+k'-1} \sigma(U^{\text{aft}}(\Gamma_p))$$

Hence

$$\sum_{p=q}^{q+k'-1} U^{\text{aft}}(\Gamma_p) + u_j > \sum_{p=q}^{q+k'-1} \left(U^{\text{aft}}(\Gamma_p) + \sigma(U^{\text{aft}}(\Gamma_p))\right)$$

As the expression $U^{\text{aft}}(\Gamma_p) + \sigma(U^{\text{aft}}(\Gamma_p))$ is lower-bounded by $2(\sqrt{2}-1)$ for any $U^{\text{aft}}(\Gamma_p) \in [0,1]$, the lemma follows. ∎

Theorem 3: A set of tasks Γ is schedulable on m processors if $U(\Gamma) \leq \left(\frac{\sqrt{17}}{3} - 1\right) 2m$.

Proof: Consider a task set Γ with $U(\Gamma) \leq \left(\frac{\sqrt{17}}{3} - 1\right) 2m$. We show that if the algorithm fails to schedule Γ, then $U(\Gamma) > \left(\frac{\sqrt{17}}{3} - 1\right) 2m$, which is a contradiction.

Assume that the algorithm declares failure at some point. Let Γ'' be the set of tasks assigned succesfully (integrally or by splitting) before the failure. From Lemmas 7 and 8 we know that all clusters successfully formed by the algorithm up to that point are utilised above $\left(\frac{\sqrt{17}}{3} - 1\right) 2$. From Lemma 9 we also know that, if the task τ_j which the algorithm attempted to split over the last candidate cluster $\{P_q, \ldots, P_m\}$ were to be assigned there (shedulability considerations aside), then the resulting utilisation of that cluster would exceed $2(\sqrt{2}-1)$. In turn, $2(\sqrt{2}-1) > \left(\frac{\sqrt{17}}{3} - 1\right) 2$.

This means that, $U(\Gamma'' \cup \{\tau_j\}) > \left(\frac{\sqrt{17}}{3} - 1\right) 2m$. Given that $\Gamma'' \cup \{\tau_j\} \subseteq \Gamma$, it then follows that $U(\Gamma) > \left(\frac{\sqrt{17}}{3} - 1\right) 2m$. ∎

4. Optimisations

A. Improving average-case performance

Since the sufficient test of Theorem 1, on which the formula (2) for sizing subtasks is based, is pessimistic, a better formula (for use inside the function split_task(τ_j)) would improve average-case performance while preserving the utilisation bound of HIME. Below, we present such a formula, combining different polynomial time-complexity schedulability tests. Theorem 4 summarises some results of interest from [12] and [13]:

Theorem 4: Let $\Gamma_p = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of background tasks scheduled by EDF on a processor P_p under interference of a high priority task $\tau_0 = (C_0, T_0)$ with $T_0 \leq T_i, i = 1, 2, \ldots, n$. No tasks miss their deadlines if $u_0 \stackrel{\text{def}}{=} C_0/T_0$ is upper-bounded by

$$\sigma(\Gamma_p, T_0) = \max\{\sigma_1(\Gamma_p, T_0), \sigma_2(\Gamma_p, T_0), \sigma_3(\Gamma_p, T_0)\} \quad (19)$$

where

$$\sigma_1(\Gamma_p, T_0) = 1 - \sum_{\tau_i \in \Gamma_p} \frac{C_i}{\left\lfloor \frac{T_i}{T_0} \right\rfloor T_0} \quad (20)$$

$$\sigma_2(\Gamma_p, T_0) = \frac{1 - U(\Gamma_p)}{1 + \frac{U(\Gamma_p)}{\left\lfloor \frac{\min_{\tau_i \in \Gamma_p}(T_i)}{T_0} \right\rfloor}} \quad (21)$$

$$\sigma_3(\Gamma_p, T_0) = \min_{\tau_i \in \Gamma_p} (\sigma'(\Gamma_p, T_i, T_0)) \quad (22)$$

where

$$\sigma'(\Gamma_p, T_i, T_0) = \begin{cases} \frac{1 - U(\Gamma_p)}{\left\lceil \frac{T_i}{T_0} \right\rceil \frac{T_0}{T_i}} & \text{if } \frac{1-U(\Gamma_p)}{\left\lceil \frac{T_i}{T_0} \right\rceil \frac{T_0}{T_i}} \leq \frac{T_i}{T_0} - \left\lfloor \frac{T_i}{T_0} \right\rfloor \\ 1 - \frac{U(\Gamma_p)}{\left\lfloor \frac{T_i}{T_0} \right\rfloor \frac{T_0}{T_i}} & \text{otherwise} \end{cases} \quad (23)$$

Proof: Eqs. (20) and (21) stem from Tests 2 and 3 of [12] (or [13]), respectively. As they establish sufficient schedulability tests, any of them can be individually used. In detail:

From Test 2 (Theorem 2) from [12] and [13], a sufficient condition for schedulability is

$$u_0 + \sum_{\tau_i \in \Gamma_p} \left(\frac{T_i}{\left\lfloor \frac{T_i}{T_0} \right\rfloor T_0} u_i \right) \leq 1 \Leftrightarrow$$

$$u_0 \leq 1 - \sum_{\tau_i \in \Gamma_p} \left(\frac{T_i}{\left\lfloor \frac{T_i}{T_0} \right\rfloor T_0} \frac{C_i}{T_i} \right) \Leftrightarrow$$

$$u_0 \leq 1 - \sum_{\tau_i \in \Gamma_p} \frac{C_i}{\left\lfloor \frac{T_i}{T_0} \right\rfloor T_0} = \sigma_1(\Gamma_p, T_0)$$

From Test 3 (Theorem 3) from [12] and [13], a sufficient condition for schedulability is

$$\left(\frac{U(\Gamma_p)}{\left\lfloor \frac{\min_{\tau_i \in \Gamma_p}(T_i)}{T_0} \right\rfloor} + 1 \right) u_0 + U(\Gamma_p) \leq 1 \Leftrightarrow$$

$$\left(\frac{U(\Gamma_p)}{\left\lfloor \frac{\min_{\tau_i \in \Gamma_p}(T_i)}{T_0} \right\rfloor} + 1 \right) u_0 \leq 1 - U(\Gamma_p) \Leftrightarrow$$

$$u_0 \leq \frac{1 - U(\Gamma_p)}{\frac{U(\Gamma_p)}{\left\lfloor \frac{\min_{\tau_i \in \Gamma_p}(T_i)}{T_0} \right\rfloor} + 1} = \sigma_2(\Gamma_p, T_0)$$

As we will now proceed to show, Equation (22) can also be used for the same purposes.

Let τ_0 and τ_i' be two tasks with their periods being T_0 and T_i and their run-times being C_0 and $C_i' = U(\Gamma_p)T_i$, respectively. According to the RM priority assignment, τ_0 has higher priority than that of τ_i'. In a critical time zone of τ_i', there are $\left\lceil \frac{T_i}{T_0} \right\rceil$ requests for τ_0. Let us now adjust C_i' to fully utilise the available processor time within the critical time zone. Two cases occur:

Case 1. The run-time C_0 is short enough that all requests for τ_0 within the critical time zone of T_i are completed before the second request for τ_i'. That is,

$$\left\lfloor \frac{T_i}{T_0} \right\rfloor T_0 + C_0 \leq T_i \Rightarrow u_0 = \frac{C_0}{T_0} \leq \frac{T_i}{T_0} - \left\lfloor \frac{T_i}{T_0} \right\rfloor$$

Thus, the largest possible value of C_i' is $C_i' \leq T_i - C_0 \left\lceil \frac{T_i}{T_0} \right\rceil$.

The corresponding processor utilisation factor is

$$U(\Gamma_p) + u_o = \quad 1 + u_o \left(1 - \frac{T_0}{T_i} \left\lceil \frac{T_i}{T_0} \right\rceil \right) \Rightarrow$$
$$u_o = \quad \frac{1-U(\Gamma_p)}{\left\lceil \frac{T_i}{T_0} \right\rceil \frac{T_0}{T_i}}$$

Case 2. The execution of the $\left\lceil \frac{T_i}{T_0} \right\rceil$th request for τ_0 overlaps the second request for τ_i'. In this case, $u_0 > \frac{T_i}{T_0} - \left\lfloor \frac{T_i}{T_0} \right\rfloor$. It follows that the largest possible value of C_i' is

$$C_i' = -C_0 \left\lfloor \frac{T_i}{T_0} \right\rfloor + T_i \left\lfloor \frac{T_i}{T_0} \right\rfloor$$

and the corresponding utilisation factor is

$$U(\Gamma_p) + u_o = \quad \frac{T_0}{T_i} \left\lfloor \frac{T_i}{T_0} \right\rfloor + u_0 \left(1 - \frac{T_0}{T_i} \left\lfloor \frac{T_i}{T_0} \right\rfloor \right) \Rightarrow$$
$$u_o = \quad 1 - \frac{U(\Gamma_p)}{\left\lfloor \frac{T_i}{T_0} \right\rfloor \frac{T_0}{T_i}}$$

The cases above correspond to what is stated in Equation (23). Therefore, the minimum processor utilisation available considering each task $\tau_i \in \Gamma_p$ yields Equation (22), as required. ■

B. Improving the least upper bound

It can be shown (see Theorem 5 below) that for algorithms constrained to no more than $\lfloor \frac{m}{2} \rfloor$ migrating tasks, the highest possible least upper bound on processor utilisation is 75%. Since HIME and other algorithms (*e.g.*, Clustered C=D [14]) are subject to this constraint, this bound also applies to it.

Theorem 5: If no more than $\lfloor m/2 \rfloor$ migrating tasks are allowed in a system with m identical processors, then the least upper bound on processor utilisation cannot exceed $0.75m$.

Proof: We construct a particular unschedulable system with an even number of processors m and with utilisation barely above $0.75m$. Let $\Gamma = \{\tau_1, \ldots, \tau_n\}$ be the set of tasks to be scheduled on m processors with $n = 1.5m + 1$ and for any $\tau_i \in \Gamma$, $u_i = 0.5 + \epsilon$. As there cannot be more than $\lfloor 0.5m \rfloor$ migrating tasks, $m + 1$ tasks have to be partitioned. This is impossible, so the system cannot be scheduled. The utilisation of this system is given by:

$$\frac{U(\Gamma)}{m} = \frac{0.5 + \epsilon}{m} (1.5m + 1) = (0.5 + \epsilon) \left(\frac{3}{2} + \frac{1}{m} \right)$$

For $\epsilon \approx 0$, $\lim_{m \to \infty} \frac{U(\Gamma)}{m} = \frac{3}{4}$, as required. ■

HIME brings about an additional constraint, namely the number of migrating tasks per processor is limited to at most one. In this section we show, however, that this additional constraint of HIME need not prevent it from matching the theoretical bound of 75%. To show this result we describe a possible configuration for HIME, modifying the way the first piece of a migrating task is dealt with.

The configuration procedure can be outlined as follows. Assume that a migrating task $\tau_s = (C_s, T_s)$ is assigned to cluster $\{P_q, \ldots, P_{q+k-1}\}$, with $k > 2$ and $\sum_{p=q}^{q+k-1} U(\Gamma_p) < 0.75k$. Call this a *spoiler task*. From Theorem 4 it is known that the lower the period of the migrating task, the higher the fraction of its utilisation that may safely be assigned to a given processor. Using this observation, we show that halving the period of the first piece of τ_s suffices to make HIME achieve at least 75% of processor utilisation. Intuitively, by "spreading out" in time the processing demand by the migrating task on that processor, this arrangement allows the latter to be better utilised. Specifically, without loss of generality, let $\tau_s^p = (C_s^p, T_s)$, $p = 1, 2, \ldots, k$, be the pieces of the spoiler task τ_s. Consider the first piece, (C_s^1, T_s). If we halve the period of just this first piece ($T_s'^1 = T_s/2$) (while leaving that of other pieces unchanged), then, after this transformation, the first piece becomes $(C_s'^1, 0.5T_s)$, with $C_s'^1$ capable of being set to at least $\frac{C_s^1 + C_s^k}{2}$ (but typically more), with no detriment to schedulability. This, in turn, may reduce the number of processors that τ_s uses from k to $k - 1$. This scenario is illustrated in Figure 5 for $k = 3$.[2] As can be seen, the first piece of τ_s is executed twice before migrating. Also, note that although this strategy generates an additional preemption on the first processor of a cluster, it may reduce the number of migrations by one due to increasing $C_s'^1$, as mentioned above.

As Figure 5 illustrates, the approach described is only applicable if the sum of execution times of all pieces of the migrating task but the first one does not exceed $\frac{T}{2}$. That is, for any spoiler task $\tau_s = (C_s, T_s)$, C_s minus what is executed by the first piece (τ_s^1) must not exceed $\frac{T_s}{2}$. Taking $C_s = u_s T_s$,

$$C_s - \sigma \left(U^{\text{aft}}(\Gamma_q), \frac{T_s}{2} \right) \frac{T_s}{2} \leqslant \frac{T_s}{2} \Rightarrow$$
$$u_s \leqslant \frac{1 + \sigma \left(U^{\text{aft}}(\Gamma_q), 0.5T_s \right)}{2} \qquad (24)$$

Next, we show that HIME achieves at least 75% of processor utilisation if the proposed modification is carried out in clusters fulfilling Condition (24). Clusters with $k = 2$ processors are addressed in Lemma 4, which establishes a lower bound of exactly 75%. For $k > 3$, we show in Lemmas 10 and 11 that this bound is greater than 75%.

Solely for better readability, since Lemmas 10 and 11 are formulated in the context of a particular cluster, we assume (without loss of generality) that the spoiler task is split over processors $P_1, \ldots, P_k$.

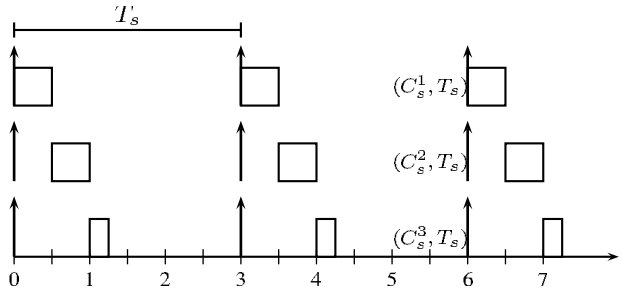
Lemma 10: Let $\tau_s = (C_s, T_s)$ be a spoiler task allocated on $k > 2$ processors by Figure 2. If Condition (24) holds and τ^1 is executed with period $T_s/2$, then $\sum_{p=1}^{k} U^{\text{aft}}(\Gamma_p) + u_s > 0.75k$ provided that $\sigma(U(\Gamma_p), T_s) \geq \frac{1 - U(\Gamma_p)}{1 + \frac{U(\Gamma_p)}{\left\lfloor \frac{T_{min}}{T_s} \right\rfloor}}$.

Proof: We first derive lower bounds on the utilisation of $U^{\text{aft}}(\Gamma_1) + \sigma(U^{\text{aft}}(\Gamma_1))$ and $U^{\text{aft}}(\Gamma_p) + \sigma(U^{\text{aft}}(\Gamma_p))$, $p = 2, \ldots, k-1$. Then, we compute the least upper bound on u_s using the derived lower bounds.

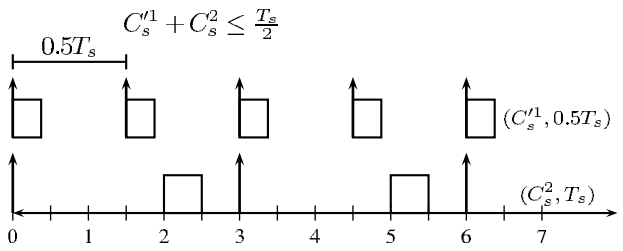
By Function split_task(), we know that

$$U^{\text{aft}}(\Gamma_p) + u_s^p = U^{\text{aft}}(\Gamma_p) + \sigma(U^{\text{aft}}(\Gamma_p), T_s), \; p = 1, \ldots, k-1$$

[2] From the proof of Theorem 3 it is known that HIME achieves a least upper bound lower than 75% only in clusters of $k = 3$ processors. In this section we present the approach for $k \geq 3$ for the sake of generalisation.



(a) Executing $\tau_s = (C_s, T_s)$ on three processors, where $C_s = C_s^1 + C_s^2 + C_s^3$.



(b) Executing τ_s on two processors by halving the period of the first sub-task subject to: $C_s'^1 + C_s^2 \leq \frac{T_s}{2}$, where $C_s = 2C_s'^1 + C_s^2$.

Fig. 5. Dealing with spoiler task $\tau_s = (C_s, T_s)$ in HIME so that its least upper bound is maximised.

Let $x = \left\lfloor \frac{T_{min}}{T_s} \right\rfloor$, where T_{min} is the minimum period of tasks in $\cup_{p=1}^{k} \Gamma_p$. By the fact that $\sigma(U^{\text{aft}}(\Gamma_p), T_s) \geq \alpha(U^{\text{aft}}(\Gamma_p))$,

$$U^{\text{aft}}(\Gamma_p) + u_s^p \geq U^{\text{aft}}(\Gamma_p) + \frac{1 - U^{\text{aft}}(\Gamma_p)}{1 + \frac{U^{\text{aft}}(\Gamma_p)}{x}} \tag{25}$$

Minimizing the right hand side of Inequality (25) yields

$$U^{\text{aft}}(\Gamma_p) + u_s^p \geq 2\sqrt{x^2 + x} - 2x \tag{26}$$

As $T_{min} \geq T_s$ and τ_s^1 is executed with period $\frac{T_s}{2}$, (26) gives $U^{\text{aft}}(\Gamma_1) + u_s^1 \geq 2\sqrt{6} - 4$ and $U^{\text{aft}}(\Gamma_p) + u_s^p \geq 2\sqrt{2} - 2$ for $p = 2, \ldots, k-1$. We now need a lower bound on $U^{\text{aft}}(\Gamma_k)$.

As function select_processors_for_next_cluster() selects processors in non-decreasing order of $U^{\text{bef}}(\Gamma_p)$, we can safely state that $U^{\text{bef}}(\Gamma_k) \geq U^{\text{bef}}(\Gamma_p),\ p \leq k$. Therefore, using the result from Lemma 6 we have that $U^{\text{bef}}(\Gamma_k) \geq U^{\text{bef}}(\Gamma_{k-1}) \geq 0.5(\sqrt{17} - 3)$. This implies that $U^{\text{aft}}(\Gamma_k) \geq 0.5(\sqrt{17} - 3)$. Conservatively assuming that $u_s^k \approx 0$,

$$U^{\text{aft}}(\Gamma_k) + u_s^k > 0.5(\sqrt{17} - 3) \tag{27}$$

Now, to compute a lower bound on the utilisation of the entire cluster, we have

$$\frac{\sum_{p=1}^{k} U^{\text{aft}}(\Gamma_p) + u_s}{k} > \frac{2\sqrt{6} - 4 + (k-2)(2\sqrt{2} - 2) + 0.5\left(\sqrt{17} - 3\right)}{k} \tag{28}$$

Since the above equation is increasing on k, the least upper bound occurs when $k = 3$, implying that

$$\frac{\sum_{p=1}^{k} U^{\text{aft}}(\Gamma_p) + u_s}{k} > \frac{\sqrt{2} + 2\sqrt{6} + 0.5(\sqrt{17} - 3) - 6}{3} \approx 0.762$$

■

Lemma 11: Let $\tau_s = (C_s, T_s)$ be a migrating task allocated on $k > 2$ processors by the algorithm of Figure 2. If Condition (24) does not hold, then $\sum_{p=1}^{k} U^{\text{aft}}(\Gamma_p) + u_s > 0.75k$ provided $\sigma(U(\Gamma_p), T_s) \geq \frac{1 - U(\Gamma_p)}{1 + \frac{U(\Gamma_p)}{\left\lfloor \frac{T_{min}}{T_s} \right\rfloor}}$.

Proof: Using α as a lower bound on σ and from the fact that Condition (24) does not hold,

$$u_s > \frac{1 + \frac{1 - U^{\text{aft}}(\Gamma_1))}{1 + \frac{U^{\text{aft}}(\Gamma_1)}{2}}}{2} = \frac{4 - U^{\text{aft}}(\Gamma_1)}{2U^{\text{aft}}(\Gamma_1) + 4} \tag{29}$$

Now consider τ_i the task used by Function select_processors_for_next_cluster() (line 5 of Figure 3) and P_k the last processor selected by this function. For the case that $k < k'$, Lemma 7 already establishes that the utilisation of the cluster in consideration exceeds 75%. Therefore, it suffices to show the same for the case that $k = k'$.

Consider two complementary cases: $u_i = u_s$ and $u_i \neq u_s$. We will show that in either case, $U^{\text{aft}}(\Gamma_1) \leq U^{\text{bef}}(\Gamma_k)$ and $u_s \leq U^{\text{bef}}(\Gamma_k)$ (and proceed from there).

Case 1 ($u_i = u_s$)**:** Then, from the task ordering $u_s \leq U^{\text{bef}}(\Gamma_k)$ and from the processor indexing by order of non-decreasing $U(\Gamma_p)$ (and since $U^{\text{aft}}(\Gamma_p) = U^{\text{bef}}(\Gamma_p)$ for every processor in the cluster when there is no task swap) $U^{\text{aft}}(\Gamma_1) = U^{\text{bef}}(\Gamma_1) \leq U^{\text{bef}}(\Gamma_k)$.

Case 2 ($u_i \neq u_s$)**:** It holds that $U^{\text{aft}}(\Gamma_1) \leq U^{\text{aft}}(\Gamma_k)$ due to the processor reindexing; since $U^{\text{aft}}(\Gamma_p) \leq U^{\text{bef}}(\Gamma_p)$ for every processor in the cluster (due to $u_s \geq u_i$, from the task ordering), it then follows that $U^{\text{aft}}(\Gamma_1) \leq U^{\text{bef}}(\Gamma_k)$.

Additionally, since u_s is an element of some Γ_p before the task swap and $U^{\text{bef}}(\Gamma_k) \geq U^{\text{bef}}(\Gamma_p)$ for every Γ_p in the cluster, it follows that $U^{\text{bef}}(\Gamma_k) \geq u_s$.

Therefore, considering that, in either case, $U^{\text{bef}}(\Gamma_k) \geq u_s$ and $U^{\text{bef}}(\Gamma_k) \geq U^{\text{aft}}(\Gamma_1)$ and that the RHS of (29) is decreasing on $U^{\text{aft}}(\Gamma_1)$, it follows that

$$U^{\text{bef}}(\Gamma_k) > \frac{4 - U^{\text{bef}}(\Gamma_k)}{2U^{\text{bef}}(\Gamma_k) + 4}$$

Using V as a short-hand for $U^{\text{bef}}(\Gamma_k)$, we get:

$$V > \frac{4 - V}{2V + 4} \Rightarrow 2V^2 + 4V > 4 - V \Rightarrow 2V^2 + 5V - 4 > 0$$

Solving the above quadratic inequality gives $V > \frac{\sqrt{57} - 5}{4}$.

Using this value for $U^{\text{bef}}(\Gamma_k)$ in Equation (11) and considering Lemma 6 which states that the average utilisation of the other $k-1$ processors in the cluster is not less than $\frac{\sqrt{17} - 3}{2}$, we

find that $\frac{u_s+\sum_{p=1}^{k} U^{\text{aft}}(\Gamma_p)}{k} > 0.75$ (since for $k = k'$ it holds that $u_s + \sum_{p=1}^{k} U^{\text{aft}}(\Gamma_p) = u_i + \sum_{p=1}^{k'} U^{\text{bef}}(\Gamma_p)$). ■

Theorem 6: A sporadic task set Γ with $U(\Gamma) \leq 0.75m$ is schedulable by HIME on m identical processors provided that, on each cluster for which Condition (24) holds, the first sub-task of the migrating task is executed twice as frequently as its other subtasks and Equation (21) is used by function split_task() to size subtasks.

Proof: From Lemma 11, the clusters with $k \geq 3$ for which Condition (24) does not hold are utilised above 75% anyway. Hence, the proof is immediate from Lemmas 4 and 10. ■

C. On overcoming the drawbacks of clustering

Having at most one migrating task per processor is conceptually the arrangement closest to the simplicity of pure partitioning and is also "gentle" towards processor caches. Yet, as shown, even this small degree of migration permits the design of algorithms with a utilisation bound as high as 75% – a theoretical limit matched by HIME. Still, the clustering resulting from this arrangement introduces a form of bin-packing-related fragmentation. Namely, the last processor in each cluster may have unutilised spare capacity.

One could overcome this drawback of clustering by reusing this capacity for additional migrating tasks on those cluster-final processors, whenever necessary (*i.e.*, in an attempt to accommodate any remaining unassigned tasks). This approach would undo the clustering but would dominate (clustered) HIME, while also inheriting its utilisation bound. Yet, although we outline this design possibility, the focus of this paper is to characterise the scheduling potential of the simpler model (at most one migrating task per processor) described so far.

D. Illustration

Positive effects related to using some optimisation strategies discussed in Section 4 are now illustrated. More specifically, we focus here on applying the results of Theorem 4 for average case improvements. To do that we use the following example:

Example 2: Let Γ be a set of tasks composed of the same five tasks as in Example 1 and a sixth task $\tau_6 = (1.92, 3)$.

The allocation of the first four tasks takes place exactly as in Example 1. Also, when selecting the processors to which τ_5 should be assigned, all four processors are selected. These initial allocation steps are indicated in first three lines of Table I, repeated in Table II for reference.

However, since the task-split procedure now uses function (19), processors capacity can be better utilised. Indeed, both τ_5 and τ_6 can be split into two pieces each, forming clusters $\{\tau_3, \tau_4, \tau_5\}$ and $\{\tau_1, \tau_2, \tau_6\}$, respectively. Notice that either Equation (20) or Equation (22) informs us that both clusters can safely use 100% of the assigned processors.

TABLE II
TASK-TO-PROCESSOR ASSIGNMENT FOR EXAMPLE 2.

	allocation of $\tau_1, \tau_2, \tau_3, \tau_4$			
Processor	P_1	P_2	P_3	P_4
Allocation	$\{\tau_1\}$	$\{\tau_2\}$	$\{\tau_3\}$	$\{\tau_4\}$
$U(\Gamma_p)$	0.6800	0.6800	0.6700	0.6700
$\sigma(U(\Gamma_p))$	0.1904	0.1904	0.1976	0.1976
$\sigma(U(\Gamma_p), T_5)$	–	–	0.3300	0.3300
$\sigma(U(\Gamma_p), T_6)$	0.3200	0.3200	–	–
	allocation of τ_5 and τ_6			
Allocation	$\{\tau_1, \tau_6^1\}$	$\{\tau_2, \tau_6^2\}$	$\{\tau_3, \tau_5^3\}$	$\{\tau_4, \tau_5^4\}$
$U(\Gamma_p) + u_i^p$	1.0	1.0	1.0	1.0

5. ASSESSMENT

The semi-partitioning paradigm underlies several new scheduling algorithms. Here we compare HIME with some relevant prior work.

A. Related work

We focus on algorithms EDF-fm [1], SPA2 [7], EDF-WM [9], C=D [5], and NPS-F [4]. The former four employ task prioritisation or migration control strategies similar to HIME whereas the latter provides the best theoretical bounds on schedulability.

HIME, SPA2, EDF-WM and C=D are based on job-split task-dispatching. When a job is released, a condition is setup specifying when the job should migrate to another processor. Usually this condition is a respective execution time threshold for the migrating job on each processor. SPA2 uses Rate-Monotonic [10] and schedules migrating tasks at the top priority. Its utilisation bound is 69.3%. EDF-WM and C=D use local EDF schedulers for each processor and establish time precedence constraints among sub-tasks via virtual deadlines. In C=D a migrating task executes at the top priority on all its processors but one; hence migrating tasks may suffer interference on just one processor. Although the task allocation schemes of EDF-WM and C=D differ, they both rely on pseudo-polynomial schedulability tests. For one variant of C=D, a utilisation bound of 72.2% was proven [14]; For others [5] we only know that it cannot be higher that 75% [14].

HIME works similarly to other scheduling algorithms. For example, EDF-fm [1] also uses EDF to schedule non-migrating tasks and migrating tasks are scheduled with higher priority. Differently from HIME, though, EDF-fm aims at soft real-time and migrates tasks at job boundaries. HIME scheduling rules ressemble a clustered variant [14] of C=D [5], but with migrating tasks *always* prioritised over non-migrating ones. By comparison, under C=D (clustered or non-clustered), this does not hold on the last processor that a task migrates to. HIME and both variants of C=D also differ in the task-to-processor allocation scheme used.

NPS-F instead assigns tasks to servers (some partitioned; some split), which are scheduled with fairness. The schedule is organised into equal timeslots, synchronised across all processors. The slot size is a trade-off between theoretical system schedulability and runtime overhead: small timeslots imply higher schedulability bounds but higher scheduling overheads. If such costs are disregarded, NPS-F can be tuned for schedulability bounds arbitrarily close to 100%. In practice, the achievable system utilisation may be quite lower [3].

B. Experimental evaluation

The overall behavior of HIME against selected work is summarised in Figure 6. Each point represents an average of $1,000$ randomly generated task sets. The graphs plot the fraction of schedulable task sets for each algorithm (and respective schedulability test). For other m (not shown), the results followed similar patterns.

Only task set utilisations in the range 70%-97.5% were considered, since task sets with lower values of utilisation are all schedulable by HIME. All synthetic task sets were generated according to the procedure described by Emberson *et al.* [6], ensuring that processor utilisation follows a uniform distribution. Task periods were generated according to a log-uniform integer distribution in the interval $[10, 1000]$.

Let the factor $\beta = n/m$ be a parameter to characterise the kind of generated task set. We considered $1 < \beta \leq 2.5$ so as to prevent HIME from taking advantages of getting the whole system partitioned upon task allocation, which often happens when generated task utilisation tends to be too small. As for the values of n, were considered: $n = m+1$; $n = 2m-1$; and $n = 2.5m$. The first criterion expresses the minimum number of tasks that makes it possible to define one migrating task. The second represents the minimum number of tasks that may lead to $m - 1$ migrating tasks, which may be possible for NPS-F, EDF-WM and C=D. The third criterion is to cover scenarios with the maximum value considered for β.

HIME was compared against C=D (EDF Split then Pack(DD-MinD)) [5], EDF-WM/DP and NPS-FΩ [3], [4]. For each algorithm we chose the version that gives the best performance. Further, although SPA2 [7] is a polynomial and low-overhead scheme, it presents a lower schedulability bound since it follows Liu and Layland's utilisation bound. Partitioned EDF following the First-Fit bin-packing heuristic was considered to illustrate to what extent the semi-partitioned scheduling approaches improve system schedulability. Also, we only performed NPS-FΩ in our experiments when partitioned EDF does not find a partition for the given task set, as recommended [4]. The task allocation order for these two approaches follow the decreasing task utilisation scheme since it is known that this gives NPS-F the best performance [4].

The utilisation bound of NPS-F depends on the timeslot, which in turn is inversely proportional to an integer parameter δ. A job of a migrating task with period T migrates up to $\lceil T/\delta \rceil$ times. The experiments used $1 \leq \delta \leq 4$, corresponding to utilisation bounds of 75%-90% of m. Higher δ are not cost-effective due to the increasing task-preemption costs.

As seen in Figure 6, HIME outperforms EDF-WM and slightly trails C=D for high values of utilisation. Also, the schedulability rates of HIME and NPS-FΩ are comparable. As the overhead in NPS-F (not considered here) is often high [3] and job-based split-task dispatching offers fewer preemptions [2], the results indicate that HIME is a cost-effective approach. Note that NPS-F is the only semi-partitioned approach with provably better schedulability guarantees than HIME. For better visualisation, Table III highlights some data from Figure 6 for high utilisation values.

TABLE III
SOME DATA FROM FIGURE 6 FOR $m = 16$.

	$n = 17$		$n = 31$		$n = 40$	
	0.95	0.975	0.95	0.975	0.95	0.975
EDF-WM	0.577	0.127	0.875	0.332	0.945	0.615
NPS-FΩ($\delta = 4$)/DU	1	1	1	0.991	1	1
$C = D$	1	1	1	1	1	1
HIME	1	1	1	0.932	1	1

With respect to HIME trailing C=D (even if slightly) in terms of scheduling potential in the experiments, despite its utilisation bound, we note the following: First, although the utilisation bound of HIME (75%) is higher than the least utilisation bounds *so far* proven for any of the variants of C=D, the exact utilisation bound of C=D is still unknown. Hence, although we know that it cannot exceed 75% (for any of its variants [14]) whether or not it is in fact *less* than is still an open question. Second, HIME was compared with the best-performing configuration of C=D (EDF Split then Pack(DD-MinD)), which is non-clustered. Hence, it does not suffer from the fragmentation penalty inherent in clustering, when the scheduling capacity of the last processor in the cluster is not exhausted. Third, C=D employs an exact test when sizing subtasks instead of the polynomial schedulability test used by HIME. Finally, it is worth of notice that C=D does not dominate HIME nor *vice versa*. For example, Example 2 is not managed by C=D.

6. CONCLUSION

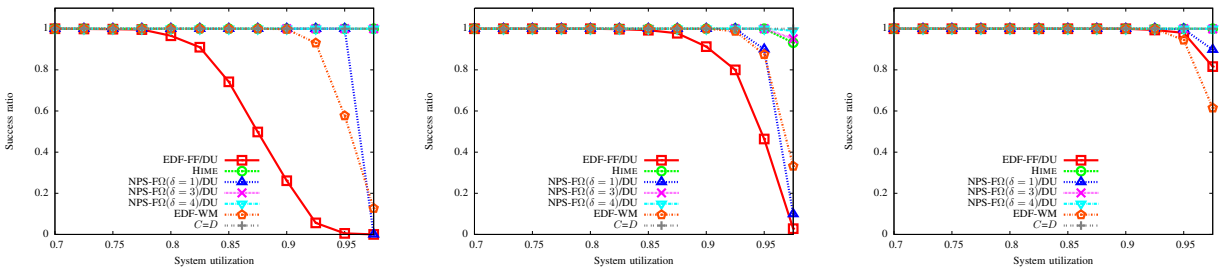
We have presented HIME, a new semi-partitioned scheduling algorithm suitable for multiprocessor real-time systems with high processor utilisation. Systems utilised up to 74.9% are guaranteed to be schedulable by HIME and an optimised variant of HIME further boosts this bound to 75%. Since 75%, as we showed, is a theoretical limit on utilisation bounds for the class of algorithms with at most m/2 migrating tasks, HIME closed one of the open problems in the literature. Our experimental evaluation of theoretical schedulability also demonstrated that HIME schedules virtually all systems utilised up to 95% or more. Given the low-overhead characteristics of semi-partitioning, we consider HIME a useful contribution to a grander vision of multiprocessor real-time systems.

Acknowledgements

This work has received financial support from the funding agencies CNPq and CAPES. This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within projects Ref. FCOMP-01-0124-FEDER-022701 (CISTER) and ref. FCOMP-01-0124-FEDER-020536 (SMARTS).

REFERENCES

[1] J. H. Anderson, V. Bud, and U. C. Devi. An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In *Proc. 17th Euromicro Conference on Real-Time Systems*, pages 199–208, 2005.



(a) $m = 16, n = 17\,(\beta \approx 1.06)$ (b) $m = 16, n = 31\,(\beta \approx 1.94)$ (c) $m = 16, n = 40\,(\beta = 2.5)$

Fig. 6. Schedulability ratio for $m = 16$ processors with $\beta = (1, 2.5]$.

[2] B. Andersson and L. M. Pinho. Implementing Multicore Real-Time Scheduling Algorithms Based on Task Splitting Using Ada 2012. In *Reliable Software Technology - Ada-Europe 2010*, volume 6106, pages 54 – 67. Springer Berlin / Heidelberg, 2010.

[3] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. Is semi-partitioned scheduling practical? In *Proc. 23rd Euromicro Conf. Real-Time Systems*, pages 125–135, 2011.

[4] K. Bletsas and B. Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. *Real-Time Systems*, 47(4):319–355, 2011.

[5] A. Burns, R. Davis, P. Wang, and F. Zhang. Partitioned EDF Scheduling for Multiprocessors using a C=D Scheme. *Real-Time Systems*, 48(1):3–33, 2011.

[6] P. Emberson, R. Stafford, and R. Davis. Techniques for the Synthesis of Multiprocessor Tasksets. In *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Syst. (WATERS'10)*, pages 6–11, 2010.

[7] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling with liu and layland's utilization bound. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 165–174, 2010.

[8] J. Jensen. Sur les Fonctions Convexes et les Inégalités entre les Valeurs Moyennes. *Acta Mathematica*, 30:175–193, 1906.

[9] S. Kato, N. Yamasaki, and Y. Ishikawa. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *Proc. 21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 249–258, 2009.

[10] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *J. of ACM*, 20(1):46–61, 1973.

[11] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4), 2011.

[12] J. A. Santos-Jr. and G. Lima. Sufficient schedulability tests for edf-scheduled real-time systems under interference of a high priority task. In *Proc. of the 2nd Brazilian Symposium on Computer Systems Engineering (SBESC)*, pages 131–136, 2012.

[13] J. A. Santos-Jr., G. Lima, and K. Bletsas. Efficient schedulability tests for real-time embedded systems with urgent routines. *Design Automation for Embedded Systems*, pages 1–20, August 2013.

[14] J. A. Santos-Jr., G. Lima, and K. Bletsas. On the processor utilisation bound of the C=D scheduling algorithm. In *Real-time systems: the past, the present, and the future (Alanfest 2013), http://www.cs.unc.edu/~baruah/AlanFest/Procs.pdf*, pages 119–132, 2013.

APPENDIX

Theorem 7 (Port of Theorem 3 from [12] [13]): Let $\Gamma_p = \{\tau_1, \ldots, \tau_n\}$ be a set of background tasks scheduled by EDF on a processor and τ_0 be the highest-priority task with $T_0 \leq \min_{\tau_i \in \Gamma_p} T_i$. There is no deadline miss provided that

$$\left(\frac{U(\Gamma_p)}{\left\lfloor \frac{\min_{\tau_i \in \Gamma_p}(T_i)}{T_0} \right\rfloor} + 1 \right) u_0 + U(\Gamma_p) \leq 1 \qquad (30)$$

Proof: Let t^{φ} be the available time for executing Γ_p within a time interval L. From Theorem 8 in [4], if (31) holds $\forall L \geq \min_{\tau_i \in \Gamma_p} T_i$, then all deadlines by Γ_p are met:

$$LU(\Gamma_p) \leq t^{\varphi} \Rightarrow U(\Gamma_p) \leq \frac{t^{\varphi}}{L} \qquad (31)$$

The minimum values of t^{φ} occur when τ_0 is periodically activated. Also, the values of L for minimizing the RHS of Equation (31) occur when the start/ending of the interval L coincide with the arrival/completion of τ_0, respectively. This is because if L further increases by ϵ, $0 < \epsilon \leq T_0 - C_0$, the value of t^{φ} is also increased by ϵ. In turn, if L decreased by a positive amount $\epsilon < C_0$, t^{φ} is kept constant. In other words, the values of L to be considered are given by $L = (k+j)T_0 + C_0$, where $k = \left\lfloor \frac{\min_{\tau_i \in \Gamma_p}(T_i)}{T_0} \right\rfloor$ and $j \in Z_+$. In this case, for each time interval of size T_0, there are $(T_0 - C_0)$ time units available for executing tasks in Γ_p, which leads to $t^{\varphi} = (k+j)(T_0 - C_0)$. Rewriting Eq. (31),

$$U(\Gamma_p) \leq \frac{(k+j)(T_0 - C_0)}{(k+j)T_0 + C_0} = \frac{(k+j)(T_0 - u_0 T)}{(k+j)T_0 + u_0 T_0} \qquad (32)$$

The right-hand side of Eq. (32) is an increasing function of j. For $j = 0$, Eq. (32) becomes Eq. (30), as required. ■

Remark 5: The expression $S^{\text{bef}} \stackrel{\text{def}}{=} 1 - (U^{\text{bef}}(\Gamma_p) + \sigma(U^{\text{bef}}(\Gamma_p)))$ is a decreasing function of $U^{\text{bef}}(\Gamma_p)$ over $[\sqrt{2} - 1,\ 1]$.

Proof:

$$\begin{aligned} S^{\text{bef}} &\stackrel{\text{def}}{=} 1 - \left(U^{\text{bef}}(\Gamma_p) + \sigma(U^{\text{bef}}(\Gamma_p))\right) \\ &= 1 - U^{\text{bef}}(\Gamma_p) - \frac{1 - U^{\text{bef}}(\Gamma_p)}{1 + U^{\text{bef}}(\Gamma_p)} \\ &= \frac{U^{\text{bef}}(\Gamma_p) - (U^{\text{bef}}(\Gamma_p))^2}{1 + U^{\text{bef}}(\Gamma_p)} \end{aligned}$$

Via first-order differentiation, we obtain

$$\frac{dS^{\text{bef}}}{dU^{\text{bef}}(\Gamma_p)} = \frac{-(U^{\text{bef}}(\Gamma_p))^2 - 2U^{\text{bef}}(\Gamma_p) + 1}{(U^{\text{bef}}(\Gamma_p) + 1)^2}$$

Since the denominator of the derivative is always positive, $\frac{dS^{\text{bef}}}{dU^{\text{bef}}(\Gamma_p)}$ is negative over the same interval that its numerator is negative. In turn, numerator is a second-order polynomial whose roots are: $-1 \pm \sqrt{2}$. Hence, it is negative over $(-\infty,\ -1 - \sqrt{2}) \cup (\sqrt{2} - 1,\ \infty)$, and so is $\frac{dS^{\text{bef}}}{dU^{\text{bef}}(\Gamma_p)}$. Therefore S^{bef} is a decreasing function of $U^{\text{bef}}(\Gamma_p)$ over $[\sqrt{2} - 1,\ 1]$. ■