A Bandwidth Reservation Strategy for Multiprocessor Real-Time Scheduling

Ernesto Massa, George Lima
Distributed Systems Lab
Department of Computer Science
UFBA - Federal University of Bahia
Salvador, Brasil
Email: ernestomassa@ufba.br, gmlima@ufba.br

***Abstract*—The problem of scheduling a set of tasks on a multiprocessor architecture is addressed. Tasks are assumed to be sporadic with arbitrary deadlines and may migrate between processors. The execution of migrating tasks is controlled by a bandwidth reservation scheme so that schedulability is guaranteed by EDF. Task migration costs are taken into consideration. Results from experiments indicate that the proposed approach performs well in terms of schedulability.**

***Keywords*-multiprocessor; scheduling; migration;**

I. Introduction

Nowadays multiprocessor architectures have become commonplace in the market. Machines equipped with chips with up to eight cores can be found in low-budget desktops and this trend will seem to continue for some time. Some experts foresee that by 2017 there will be 4096-core chips [21]. Current prototypes by Intel are built with 80 cores [17]. This scenario has motivated the real-time systems research community to develop efficient real-time scheduling mechanisms and several achievements have been made in this area.

In this paper we address this problem and give a novel scheduling mechanism for scheduling a set of independent sporadic tasks with arbitrary deadlines in a multiprocessor architecture. Tasks are statically allocated to processors but some tasks are allowed to migrate between predefined processors. In other words, task migration is controlled. Migration-control schemes have been proposed recently [1][2][5][19][20] and usually compare favorably in terms of schedulability bounds against either migration-uncontrolled [6][7][8][16] or migration-forbidden [10][14] schemes.

The proposed task allocation scheme follows similar reasoning used by recently published works [1][2]. A common time slot T is defined. Reserves of such slots are assigned to migrating tasks so that they can execute in two processors without overlapping in time. We have developed a bandwidth reservation mechanism to control the amount of execution of migrating and non-migrating tasks. We name this approach EDF-BR (EDF with Bandwith Reservation). Utilization-based schedulability tests are used to define the reserves, providing an efficient allocation mechanism. Each processor executes its assigned tasks according to EDF. Another characteristic of our approach is that task migration costs are taken into consideration. To the best of our knowledge, these costs have been neglected up to now.

This work was funded by CNPq and FAPESB.

EDF-BR is evaluated using a large number of randomly generated task sets. As will be shown, EDF-BR exhibits similar performance when compared to recently published works. Nonetheless, unlike these related approaches, EDF-BR uses polynomial-time schedulability tests, which makes it a much more efficient scheme. This characteristic is very useful for dealing with complex many-processor real-time systems.

The remainder of this paper is structured as follows. Section II summarizes related work. The system model and notation used throughout this paper are given in Section III. EDF-BR is described in Section IV and its analysis is derived in Section V. Results from simulation are presented in Section VI. Concluding remarks are drawn in Section VII.

II. Related Work

The problem of scheduling a set of real-time tasks in a multiprocessor system has extensively been studied and several approaches to this problem have been proposed. This section presents only a brief summary of them, highlighting those approaches most related to EDF-BR.

Most work in the area can be classified into two groups. There are schemes that partition the task set so that tasks are statically allocated to processors and do not migrate. Although this migration-forbidden approach is interesting because usual scheduling algorithms can be used in each processor, it presents some scheduling anomalies [3][4][15]. Also, it can be shown that some schedulable systems cannot be feasibly scheduled if task migration is forbidden [12]. Scheduling approaches that do not impose any restriction on task migration lie in the second group. Usually they are based on a global queue and greedily assign tasks to processors at runtime. Priority-based scheduling policies [6][7][8][23] can be used to order this queue. These policies usually present a low schedulability bound. Although some approaches are optimal in terms of schedulability [9][13][18], they usually present high implementation overhead or consider a restrictive task model.

Recent work addresses a different approach to the problem according to which tasks are statically allocated to processors but some of them are allowed to migrate in a controlled manner. The amount of time and the processors to execute migrating and non-migrating tasks are defined off-line. This scheme has been used by EDF-fm [16], EGK [1][5], EDF-SS[2], EDF-WM[20], among others [11]. Interesting results have been found by such a branch of work. Considering periodic [5] and sporadic tasks [1], some authors have shown that schedulability can be traded with preemptions. However, only tasks with deadlines equal to periods were considered. Arbitrary deadlines have been dealt with by using pseudo-polynomial time-demand analysis [2][20].

The approach proposed in this work follows the migration-control strategy, similarly to ones mentioned above. We consider polynomial-time utilization-based tests to allocate tasks to processors like EGK [1][5]. Nonetheless, sporadic tasks with arbitrary deadlines are taken into consideration, which makes the approach described here more efficient in terms of time complexity than EDF-SS or EDF-WM. Further, to the best of our knowledge, EDF-BR is the first multiprocessor scheduling approach that takes migration costs into account. As will be shown experimentally, migration costs have a great impact on the system schedulability and so cannot be neglected.

III. Notation and System Model

We consider a system composed of n sporadic tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be scheduled in a machine Π with m identical processors $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$. There may be infinite occurrences of the same task τ_i during the system execution, each of which is called a job of τ_i. Each task τ_i is independent of each other and is defined by the tuple $\tau_i = (C_i, D_i, T_i, \mu_i)$, where: T_i represents its minimal inter-arrival time; C_i is its worst-case execution time; D_i is its relative deadline and μ_i is its migration cost. Parameter μ_i represents, for example, extra execution time due to cache-misses when τ_i migrates from one processor to another. Although determining such costs is a current research issue, recent results have been seen [22]. In our model, parameter μ_i is used as an attempt to incorporate the effect of migration for schedule purpose.

We conservatively require that each task τ_i finish its execution within $\Delta_i = \min(D_i, T_i)$. The processor demand of τ_i within this execution window is then defined as $\delta_i = C_i/\Delta_i$. Despite this conservative approach, experiments have shown similar results in terms of schedulability when comparing with related work, as will be seen in Section V.

A schedule Γ is a function that associates a task in Γ to a processor in Π over time. We consider only *valid* schedules, according to which: at any time t a processor executes, at most, one job and the same job cannot be executed in more than one processor; and no job is executed more than once. A valid schedule is *feasible* if no job misses its deadline in such a schedule. A system is `schedulable` if there is some feasible schedule for it.

Considering a given system composed of a set of tasks Γ and a multiprocessor machine Π, in this work we derive a scheduling strategy to produce a feasible schedule for it. The proposed approach is based on off-line processor bandwidth reservations. Each reserve is carried out via scheduling servers. More specifically, a task $\tau_i \in \Gamma$ executes on processor $\pi_x \in \Pi$ through a server $S_{i,x}$. Server $S_{i,x}$ is defined by a tuple $(Q_{i,x}, D_{i,x}, T_{i,x}, \text{type})$, meaning that τ_i has the right to use $Q_{i,x}$ time units within $T_{i,x}$ and its execution must finish within $D_{i,x}$, the server deadline. The value $Q_{i,x}$ is called the capacity of $S_{i,x}$. The system controls the current budget of $S_{i,x}$, denoted $q_{i,x}$, which assumes values between 0 and $Q_{i,x}$. A server can only be scheduled for execution if its budget is not null. The set of the defined servers is denoted Ω, which are scheduled according to EDF on each processor $\pi_x \in \Pi$. In other words, the server with the earliest absolute deadline in π_x is chosen to execute as long as it has positive budget and its associated task is waiting for execution.

There are three types of server in Ω. If there is enough bandwidth to successfully execute a task τ_i in π_x, an *ordinary* server is defined in Ω and is associated to the execution of τ_i. Otherwise, τ_i must migrate between two processors. These servers, named *primary* and *secondary*, are associated to the execution of migrating tasks. Migration costs are taken into consideration when defining these servers. Primary and secondary servers are allocated to two distinct processors. The rules to create Ω and schedule these servers will be defined in the next section.

IV. The Proposed Strategy

Before describing the procedure to define the servers in Ω (Section IV-C) and the rules that drive their behavior (Section IV-D), we motivate the proposed strategy in Section IV-A and give an overview of it in Section IV-B.

A. Illustration and Motivation

In order to illustrate the EDF-BR (*EDF with Bandwidth Reservation*) strategy, consider the following illustrative example.

Example IV.1. *Let $\Gamma = \{\tau_1, \tau_2, \tau_3\}$ be a periodic task set to be scheduled in two processors $\Pi = \{\pi_1, \pi_2\}$. Assume that $C_1 = 3$, $C_2 = 1.5$, $T_1 = T_2 = 4$, $C_3 = 6$ and $T_3 = 8$. Also, consider that $D_i = T_i$, $i = 1, 2, 3$ and $\mu_1 = \mu_2 = \mu_3 = 0.5$. All tasks arrive at time $t = 0$. Figure 1(a) shows a global EDF schedule for this system, where τ_3 misses its deadline and π_1 is underutilized. However, as can be seen in Figure 1(b), there is a feasible schedule for this system.*

It may be noted in Figure 1(b) that the execution of τ_2 is split into two parts within each window $T = 4$, corresponding to servers $S_{2,1} = (1, 1, 4, \text{`sec'})$ and $S_{2,2} = (1, 1, 4, \text{`pri'})$, which are scheduled so that there is no overlapping between their executions. While $S_{2,1}$ takes care of the second execution part allocated to π_1, $S_{2,2}$ is responsible for the first one. As there is a cost of 0.5 time units associated with the migration of τ_2, its total execution

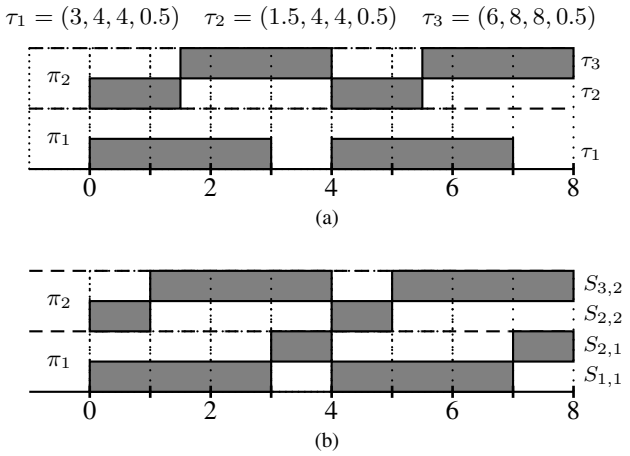


Figure 1. Illustrative example: (a) EDF; (b) EDF-BR.

cost will be 2 time units, as can be seen in Figure 1(b). According to EDF-BR, the servers can only execute when they have positive budgets. For this example, this is true for the primary server $S_{2,2}$ in intervals $[0, 1]$ and $[4, 5]$ while the corresponding secondary server has positive budgets in $[3, 4]$ and $[7, 8]$.

Tasks τ_1 and τ_3 are scheduled via ordinary servers, namely $S_{1,1} = (3, 4, 4,$ 'ord') and $S_{3,2} = (6, 8, 8,$ 'ord'). In the figure, the former server has positive budgets in $[0, 3]$ and $[4, 7]$ while the budget of the latter is positive during $[0, 8]$. As the servers are scheduled according to EDF and ties are broken in favor of primary and secondary servers, it is guaranteed that no ordinary server interferes in the execution of both primary or secondary servers. The rules that define the schedule will be detailed shortly. Next, we give some intuition on the bandwidth allocation procedure.

B. Outline

The definition of Ω can be summarized as follows:

1) Choose a common time window length $T \leq \Delta_i, i = 1, \ldots, n$ to be used for the servers of all migrating tasks. In Figure 1(b), $T = 4$. This approach is used to synchronize their executions in *time slots* so that the executions of servers associated to the same task do not overlap in time. Select π_x to start allocating servers and go to step 2.
2) Following a non-increasing order of processor demand, select each task $\tau_i \in \Gamma$ for which there is no allocated server. If a task can be assigned to an ordinary server in the current processor π_x, it is done. Otherwise, go to step 3. As can be seen in Figure 1(b), in this step τ_1 and τ_3 are assigned respectively to $S_{1,1}$ and $S_{3,2}$.
3) Calculate the largest budget still available for a secondary server in π_x and select a task τ_i which will suffer the smallest penalty as for migration.
4) Create secondary and primary servers, allocated to processors π_x and π_{x+1}, respectively. The latter server has capacity $Q_{i,x+1}$ while the capacity of the former is $Q_{i,x}$ so that τ_i is completely served on both processors and the execution time as for μ_i is granted. If this is not possible, a feasible schedule cannot be found by the proposed approach. The case of τ_2 in Figure 1(b) illustrates how two servers associated to the same tasks (with $\mu_i = 0.5$) are defined. The secondary server uses the remaining 25% of π_1 and the primary server is enough to successufuly complete the execution of τ_2 taking into account the migration cost μ_2.
5) Go to the next processor and repeat steps 2-5 until there is no task left to be served or no processor available to be allocated. In the latter case, a feasible schedule cannot be found by the proposed strategy.

The next two sections detail the scheme outlined above and define how servers are managed at runtime.

C. Processor Bandwidth Allocation

Algorithm 1 is actually a detailed version of the allocation strategy previously outlined. All primary and secondary servers are created with periods equal to T, an input parameter of the algorithm, which defines *time slots* for synchronizing the execution of the servers. If a task τ_i needs to migrate between two processors, with a migration cost μ_i, it will be served by a pair of servers, one primary and one secondary. Since τ_i must finish within Δ_i and all primary and secondary servers have period T, τ_i must finish within $T\lfloor \Delta_i/T \rfloor \leq \Delta_i$, and will execute

$$Q_i = \frac{C_i}{\lfloor \Delta_i/T \rfloor} + \mu_i \quad (1)$$

in each *time slot*. The value of Q_i is computed in line 1 for all tasks and is used later in the algorithm to define the capacities of primary and secondary servers.

It is worth emphasizing that the processor bandwidth allocation procedure tries to completely fill a processor with non-migrating tasks. When this is not possible, a primary and a secondary server are created to serve the task that migrates between the current and the next processors.

When an ordinary server $S_{i,x}$ is allocated to π_x, there may already be a primary (but no secondary) server in π_x. Let the capacity of this primary server be Q_x^p, $0 \leq Q_x^p < \Delta_i$. Recall that primary servers must execute at the beginning of each slot of size T. Also, its execution cannot suffer interference of ordinary servers, which is achieved by shortening the deadline of ordinary servers to a time before the execution of primary servers. This will be better explained in Section IV-D. Since the amount of time a deadline is shortened is bound by the capacity of a primary server, ordinary server demands can be increased to $\delta_i' = C_i/(\Delta_i - Q_x^p)$ (line 7). Note that at the time ordinary servers are being allocated to π_x, no secondary server has been allocated to π_x yet. If there is enough processor capacity to serve demand δ_i', an ordinary server is created to serve this task with $Q_{i,x} = C_i$ and $D_{i,x} = T_{i,x} = \Delta_i$ (lines 8-10). Variable δ accounts for the sum of all ordinary server

Algorithm 1: Server Allocation Procedure

Input: Set of n tasks Γ
Input: Set of m processors Π
Input: $T \in (0, \min(T_1, \ldots, T_n, D_1, \ldots, D_n)]$
Output: Ω if feasible; $\emptyset$ otherwise

1 $Q_i \leftarrow C_i / \lfloor \Delta_i / T \rfloor + \mu_i$, for all $\tau_i \in \Gamma$
2 Sort Γ such that $\frac{C_i}{\Delta_i} \geq \frac{C_{i+1}}{\Delta_{i+1}}, i = 1, \ldots, n-1$
3 $Q_x^p \leftarrow 0$; $Q_x^s \leftarrow 0, x = 1, \ldots, m$
4 $x \leftarrow 1$; $\delta \leftarrow 0$; $\Omega \leftarrow \emptyset$
5 **while** $x \leq m \wedge \Gamma \neq \emptyset$ **do**
6 **for** $\tau_i \in \Gamma$ **do**
7 $\delta_i' \leftarrow C_i / (\Delta_i - Q_x^p)$
8 **if** $\delta_i' \leq 1 - (Q_x^p / T + \delta)$ **then**
9 $\Gamma \leftarrow \Gamma / \{\tau_i\}$
10 $\Omega \leftarrow \Omega \cup \{S_{i,x} = (C_i, \Delta_i, \Delta_i, \text{`ord'})\}$
11 $\delta \leftarrow \delta + \delta_i'$
12 **if** $x = m \wedge \Gamma \neq \emptyset$ **then return** $\emptyset$
13 $Q_x^s \leftarrow \arg\max_{Q \in \mathbb{R}} \{\frac{Q + Q_x^p}{T} + \sum_{S_{i,x} \in \Omega} \frac{Q_{i,x}}{\Delta_i - \max(Q, Q_x^p)} \leq 1\}$
14 $\tau_i \leftarrow \arg\min_{\tau_j \in \Gamma} \{\frac{Q_j}{T} - \frac{C_j}{\Delta_j} | Q_j \leq T \wedge \mu_j < Q_x^s\}$
15 **if** τ_i *is found* **then**
16 $\Omega \leftarrow \Omega \cup \{S_{i,x} = (Q_x^s, Q_x^s, T, \text{`sec'})\}$
17 $Q_{x+1}^p \leftarrow Q_i - Q_x^s$
18 $\Omega \leftarrow \Omega \cup \{S_{i,x+1} = (Q_{x+1}^p, Q_{x+1}^p, T, \text{`pri'})\}$
19 $\Gamma \leftarrow \Gamma / \{\tau_i\}$
20 $x \leftarrow x + 1$
21 $\delta \leftarrow 0$
22 **if** $\Gamma \neq \emptyset$ **then return** $\emptyset$
23 **else return** Ω

demands already allocated to π_x. During the bandwidth allocation procedure the server deadline is Δ_i. The absolute server deadline is shortened at runtime but only if there is a need to do so, as will be explained in Section IV-D.

When no other ordinary server can be allocated to a processor π_x, its remaining processor capacity can be used to define a secondary server. In this case, another processor (π_{x+1}) is needed since a primary server will be defined. Hence, if $x = m$ and there are still tasks with no allocated server, the algorithm fails to define Ω (line 12). Otherwise, the algorithm goes on defining secondary and primary servers.

In order to define a secondary server, the algorithm searches for a not yet served task $\tau_i \in \Gamma$ so that the remaining capacity in π_x is best used. This is carried out in lines 13-14. There are two aspects worth noticing in these lines. First, as a secondary server will be defined in π_x, it is necessary to account for the effect of deadline shortening, as mentioned before. Nonetheless, now the capacities of both primary and secondary servers must be considered. Indeed, the deadline of ordinary servers can be shortened by at most the maximum of the capacities of primary and secondary servers. Second, when determining which task will migrate in line 14, its migration cost is taken into account. As can be seen, if the migration cost is higher than the capacity left on π_x or the total execution cost plus migration is greater than the time slot, it is better to allocate an ordinary server than to force such a task to migrate. If some τ_i is selected to migrate, a secondary server $S_{i,x}$ is created, with $Q_{i,x} = D_{i,x} = Q_x^s$ and $T_{i,x} = T$, filling up the current processor. Its primary server $S_{i,x+1}$, with $Q_{i,x+1} = D_{i,x+1} = Q_{x+1}^p$ and $T_{i,x+1} = T$ is defined to complete the total execution cost of τ_i (Q_i) within a time slot T (lines 15-18).

It is interesting to observe that the chosen value of T has some impacts on both the schedule feasibility and on the number of preemptions. Considering Example IV.1 as illustration, if one chose $T = 3$ instead of $T = 4$, a feasible schedule would not be possible. Indeed, in this case, Algorithm 1 would generate $S_{1,1} = (3, 4, 4, \text{`ord'})$, $S_{2,1} = (0.458, 0.458, 3, \text{`sec'})$, $S_{2,2} = (1.542, 1.542, 3, \text{`pri'})$ and no server for τ_3 could be generated. In this case it is easy to see that the increased demand of ordinary server $S_{1,1}$ in π_1 would be $\frac{3}{4-0.458} = 0.847$, the demand of secondary server $S_{1,2}$ in π_1 would equal $\frac{0.458}{3} = 0.153$ and the demand of primary server $S_{2,2}$ in π_2 would be $\frac{1.542}{3} = 0.514$. The increased demand of server $S_{2,3}$ blocked by a budget of 1.542 would be $\frac{6}{8-1.542} = 0.929$ beyond what was available in π_2.

Also, note that the lower the value of T, the higher the number of preemptions of the migratory tasks. These observations suggest that if T assumes the highest value so that $T_i - T\lfloor \Delta_i / T \rfloor$ is minimized, one can get good trade-offs for schedulability and preemption, an aspect also observed in other approaches [1][5]. Clearly, choosing $T = \text{gcd}(\Delta_1, \ldots, \Delta_n)$ makes $T_i - T\lfloor \Delta_i / T \rfloor = 0$ for all τ_i. However, if task periods are co-prime, this strategy may imply high preemption costs.[1]

D. Scheduling Policy

At any time-instant, each server $S_{i,x} = (Q_{i,x}, D_{i,x}, T_{i,x}, type) \in \Omega$ has a budget, $q_{i,x}$, $0 \leq q_{i,x} \leq Q_{i,x}$. The budget is consumed and replenished during the system execution according to some consumption and replenishment rules, respectively. These rules determine the behavior of the servers in Ω. Before detailing such rules, let us define the server states.

1) Server States: At any time a server can be in one of the following four states:

Ready. In this state a server can be scheduled for execution since it has both some task waiting for execution and a positive budget.
Waiting. There is no pending task to be served. Hence, a server cannot be scheduled for execution while it is in this state.

[1]An interesting compromising strategy could be dividing Γ into subsets, allocating each subset to a group of processors so that the relations between task periods can be explored. In this paper we do not attempt to do so since we focus here on the scheduling strategy alone.

Backlogged. There is some pending task to be served but its current budget is exhausted. Therefore, it cannot be selected for execution.
Executing. If a ready server is selected for execution, it is in the executing state. At any time there is up to one server in this state per processor.

The transitions between server states are illustrated in Figure 2. If $S_{i,x}$ is in the ready state, it goes to the executing state when it is selected by the dispatcher. Once executing, $S_{i,x}$ may go to the other three states depending on its budget and the task it is serving, as indicated by the arrows in the figure. For example, if $S_{i,x}$ is executing and its budget is depleted before its executing task finishes, the server stays backlogged until its budget is replenished. Nonetheless, if there is no task to be served, $S_{i,x}$ is put in the waiting state until a new job arrives. If $q_{i,x} > 0$ at this time, $S_{i,x}$ becomes ready. Otherwise, it stays backlogged until $q_{i,x}$ is replenished.

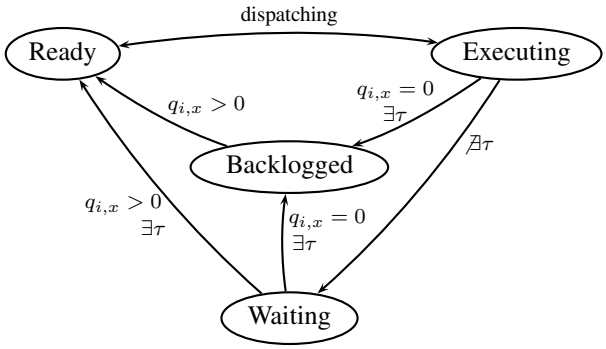


Figure 2. Server states and their transitions.

2) Server Maintenance and Scheduling Rules: There are four types of rule that drive the behavior of the servers and their scheduling.

1) Replenishment rules. The budget $q_{i,x}$ of any server $S_{i,x} \in \Omega$ is replenished to the corresponding server capacity $Q_{i,x}$. The replenishment times depend on the server type:
   a) The budgets of ordinary servers are replenished upon the arrival of the task they are serving. Hence, by the task model, the minimum time between replenishment is known.
   b) Primary servers. The replenishment operation takes place at the beginning of each time slot, i.e. kT, $k = 0, 1, \ldots$.
   c) Secondary servers. $S_{i,x}$ is replenished at times $kT - Q_{i,x}$, $k = 1, 2, \ldots$. It should be noted that $Q_{i,x}$ gives the necessary time to execute $Q_{i,x}$ time units on processor π_x at the end of each *time slot*.
2) Consumption rules. A budget $q_{i,x}$ of a server $S_{i,x}$ is consumed in two different ways, depending on the server type. Whenever being consumed, their values are decreased at the rate of 1 per time unit.
   a) If $S_{i,x}$ is an ordinary server, $q_{i,x}$ is consumed when the server is in the executing state. Otherwise, $q_{i,x}$ is kept unchanged.
   b) If $S_{i,x}$ is either a primary or a secondary server, $q_{i,x}$ is consumed whenever $q_{i,x} > 0$ independent of the server state.
3) Deadline shortening rule. Let t be the arrival time of a task τ_i served by an ordinary server $S_{i,x}$. Let S be a primary or a secondary server allocated to π_x with absolute deadline and replenishment time equal to d and r, respectively. If $r < t + D_{i,x} < d$, the deadline of $S_{i,x}$ is altered to r.
4) Scheduling rule. Servers are scheduled by EDF. If ordinary servers have the same deadline as primary or secondary servers, the scheduler must select the latter in detriment of the former.

In order to implement 1b-1c, timers are needed. For 1b, a single (global) timer can be defined for the primary servers with period T. Implementing 1c requires a timer per secondary server, i.e, at most one timer per processor. Rule 1a is event-triggered, which can be implemented/triggered by the same interrupt handler that deals with the task activation.

It is important to emphasize that due to the server replenishment and consumption rules, the execution of a primary server takes place at the beginning of the time slot while a secondary server can execute only at the end of this time slot. Rules 3 and 4 ensure that no ordinary server will interfere in the execution of a primary or secondary server, as will be seen in Lemma V.2.

The following example illustrates a particular scenario showing how the task set presented in Example IV.1 is scheduled.

Example IV.2. *Consider the task set given in Example IV.1. The first jobs of tasks τ_1, τ_2 and τ_3 arrive at time instants 3, 2 and 1, respectively, as indicated by the up-arrows in Figure 3. Then the other jobs arrive periodically. Job deadlines are represented by down-arrows. The figure shows the server budgets along the time line. The chosen time slot is $T = 4$.*

As can be observed in the example, π_1 and π_2 are idle until time 3 and 1, respectively. Nonetheless, the budget $q_{2,2}$, of the primary server $S_{2,2}$, is consumed from the beginning. When τ_2 arrives, at time 2, this budget is null. On the other hand, the budget of ordinary servers is kept unchanged when the servers are not executing. Task τ_2 starts to be served at time 3 by $S_{2,1}$, a secondary server. Task τ_1 is not served in the first time slot because $S_{1,1}$ is preempted by $S_{2,1}$ until time 4. At this time, the primary server is replenished. Then $S_{1,1}$ and $S_{2,2}$ start executing since they are the earliest deadline servers in the ready state on π_1 and π_2, respectively. As can be seen, all tasks meet their deadlines.

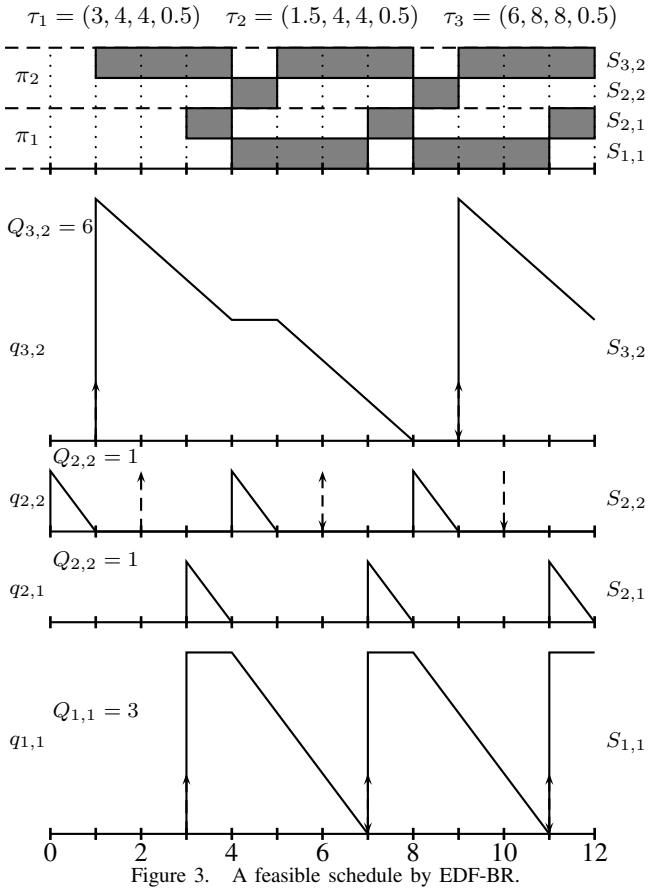


Figure 3. A feasible schedule by EDF-BR.

V. Feasibility Analysis

In this section we show that the proposed approach finds a feasible schedule for a system whenever Algorithm 1 terminates successfully. Hereafter we assume that Ω is the set of servers defined by Algorithm 1, T is the time slot used to define Ω for some task set Γ and a multiprocessor Π. The servers in Ω follow the rules stated in Section IV-D. We start showing in Lemmas V.1 and V.2 that the execution of primary and secondary servers do not suffer interference of any server.

Lemma V.1. *The executions of primary and secondary servers associated to the same processor or to the same task do not overlap in time.*

Proof: If there is no primary or secondary servers in Ω, the lemma trivially holds. Otherwise, since by the algorithm such servers are created pairwise, there must be at least a pair of primary and secondary servers in Ω. Let S^p and S^s be any pair of primary and secondary servers with budgets Q^p and Q^s, respectively. Consider an integer $k \geq 0$. S^p and S^s are replenished at times kT and $(k+1)T - Q^s$ by rules 1b and 1c, respectively. By rules 2a and 2b, their budgets are exhausted at times $kT + Q^p$ and kT, respectively. Thus, S^p and S^s can only be scheduled for execution at the beginning and ending of each time slot and their executions can not overlap in different time slots. Assume by contradiction that their executions overlap within a time slot. By the replenishment and consumption rules, this means that $kT + Q^p > (k+1)T - Q^s$, which implies that

$$Q^p + Q^s > T \tag{2}$$

There are two cases to be considered, either S^p and S^s serve the same task τ_i or they are allocated to the same processor, π_x, say. For the former case, by line 17 of Algorithm 1, $Q_i = Q^s + Q^p$ and from line 14 $Q_i \leq T$, which contradicts (2).

Now consider the case where S^p and S^s are allocated to a processor π_x. Q^s is computed by line 13 of Algorithm 1 so that

$$\frac{Q^s + Q^p}{T} + \underbrace{\sum_{S_{i,x} \in \Omega} \frac{Q_{i,x}}{\Delta_i - \max(Q^s, Q^p)}}_{\geq 0} \leq 1 \tag{3}$$

As $\Delta_i > \max(Q^s, Q^p)$, (3) implies that $Q^s + Q^p \leq T$, which also contradicts (2). ■

Lemma V.2. *Ordinary servers do not interfere in the execution of primary and secondary servers.*

Proof: Relative deadlines of ordinary servers cannot be less than $T \leq \min_{\tau_i \in \Gamma}(\Delta_i)$ while primary and secondary servers have their relative deadlines equal to their capacities, which are less than T. Also, rule 3 ensures that the absolute deadlines of ordinary servers are not less than those of current primary and secondary servers. As all servers are scheduled according to EDF and ties are broken in favor of primary and secondary servers (by rule 4), the lemma follows. ■

We will show now that all servers in Ω meet their deadlines.

Lemma V.3. *The set of servers Ω is schedulable by EDF.*

Proof: It follows from Lemmas V.1 and V.2 that primary and secondary servers in the executing state do not suffer any interference and so they meet their deadlines. Also, as the generated schedule follows EDF ordering, in order to show that ordinary servers meet their deadlines it is enough to show that the demand of the servers allocated to a given processor π_x does not exceed 100%.

Let S^p and S^s be the primary and secondary servers allocated to π_x and assume that their capacities are Q^p and Q^s, respectively. Due to the replenishment and consumption rules and Lemmas V.1 and V.2, these servers demand $(Q^p + Q^s)/T$ of π_x. As the deadlines of ordinary servers $S_{i,x}$ are set to $\Delta_i - \max(Q^s, Q^p)$ in the worst case, the maximum demand in π_x is given by

$$\frac{Q^s + Q^p}{T} + \sum_{S_{i,x} \in \Omega} \frac{Q_{i,x}}{\Delta_i - \max(Q^s, Q^p)},$$

which is not greater than 1 by Algorithm 1 (line 13). ■

Now, we will show that a system is feasibly scheduled by the proposed approach if Algorithm 1 generates Ω.

Theorem V.1. *Let Ω be a set of servers generated by Algorithm 1 for a given time slot and a task set Γ. No task misses its deadline if its associated server(s) in Ω is(are) scheduled by EDF.*

Proof: Without loss of generality, let us focus on a job of some task τ_i released at time instant t. Recall that we require that any task τ_i finish executing within $\Delta_i = \min(D_i, T_i)$. Since no other job of this task is being considered, we refer to this job as τ_i for the sake of notation simplicity.

Two cases must be considered. Assume first that τ_i is served by an ordinary server S allocated to some $\pi_x \in \Pi$. Note that the budget of S equals C_i, S meets its deadline (by Lemma V.3) and $t + \Delta_i$ is not less than the deadline of S. Thus, τ_i is guaranteed to finish its execution by its deadline.

Now, consider that τ_i is served by a pair of primary and secondary servers, S^p and S^s, respectively. There are $\eta_i = \lfloor \Delta_i / T \rfloor$ instances of such servers all of which meet their deadlines by Lemma V.3. Also, the deadlines of the last instance of these servers are not greater than $t+\Delta_i$. Further, there is no overlapping between S^p and S^s by Lemma V.1. Hence they serve $\eta_i(Q^p + Q^s) = C_i + \eta_i \mu_i$ (by line 1) time units within Δ_i, which suffices to successfully finish τ_i, taking into account its migration. Therefore, τ_i cannot miss its deadline for the second case either. ■

VI. Assessment

In order to carry out the assessment, synthetic task sets were randomly generated. Both constrained and arbitrary deadlines were considered. The evaluation metric was the observed percentage of task sets guaranteed to be schedulable. Systems with $m = 4$,8 and 16 identical processors were considered.

Each task set Γ used in the evaluation was generated according to the following procedure. First, the utilization of Γ was defined to be equal to a given value in the interval $[0.65, 1.00]$. Then task periods were generated according to a uniform distribution in the interval $[100, 3000]$ time units. Task utilizations $u_i = C_i/T_i$ were generated in a given interval $[u, 1.0]$, where u was equal to either 0.1 or 0.5. After generating the values of u_i and T_i, the values of C_i were obtained. Task deadlines were randomly generated so that either $C_i < D_i \leq T_i$ (for constrained deadlines) or $C_i < D_i \leq 2T_i - C_i$ (for arbitrary deadlines). As a means of selecting systems that are likely to be schedulable we discarded those task sets whose total demand $\sum_{\tau_i \in \Gamma} \delta_i$ was greater than $1.2m$.

The results of the experiments will be presented in the following two sections. First we compare the performance of EDF-BR with those obtained by related work. Then we evaluate the effect of migration cost on the performance of EDF-BR. Most results will be presented in graphs. Each point in each graph corresponds to the average for $1,000$ task sets.

A. Comparison Results

EDF-BR was evaluated by comparing it with EDF-SS [2] and EDF-WM [20], two recent related approaches. Like EDF-BR, EDF-SS makes use of a time slot parameter T, which is set to the minimum of the task periods divided by a factor. We used the factor equal to 4, as it was recommended [1] and used in other experimental evaluations [20]. Then the same resulting time slot value was used to configure EDF-BR. Also, the version of EDF-WM (sorted) with better performance was chosen [20]. As these approaches do not take migration costs into consideration, we set $\mu_i = 0$ for all tasks.

Figures 4, 5 and 6 present the results. As can be seen, in general, EDF-BR performs very similarly to EDF-WM. EDF-SS has a better performance when task sets with low utilization tasks are considered or when the number of processors is low. In the other scenarios, its rejection rate tends to be higher than those found by EDF-BR and EDF-WM. Although these three approaches have similar overall performance, it is important to emphasize that EDF-BR uses polynomial-time utilization-based tests while the other two are based on pseudo-polynomial time-demand analysis. It can be seen by Table I that the task allocation process by EDF-BR runs much faster than the other two approaches, and the difference of time-demand grows as the number of processors increases.

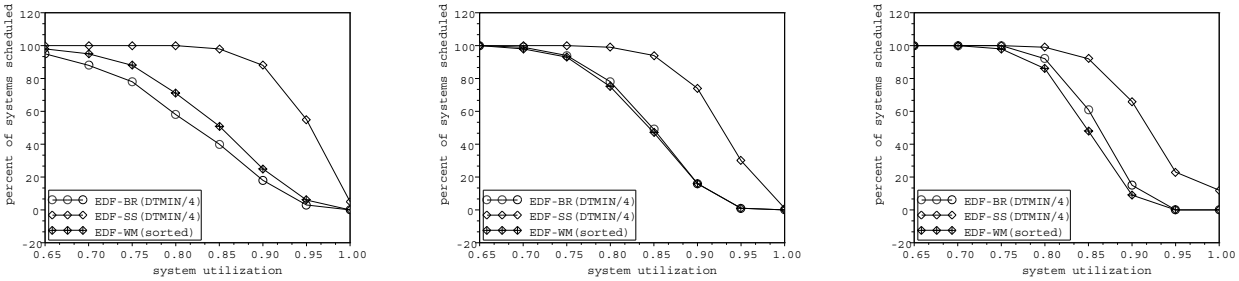
B. Migration Effects

The effect of task migration was measured. For each generated task τ_i, a migration cost μ_i was defined according to a uniform distribution in the interval $[0, \alpha C_i]$. Four values of α were used 0%, 1%, 5% and 10%. As can be seen in Figure 7, migration costs have a considerable impact on the system schedulability. This illustrates the importance of considering such costs into the scheduling mechanisms.

VII. Conclusion

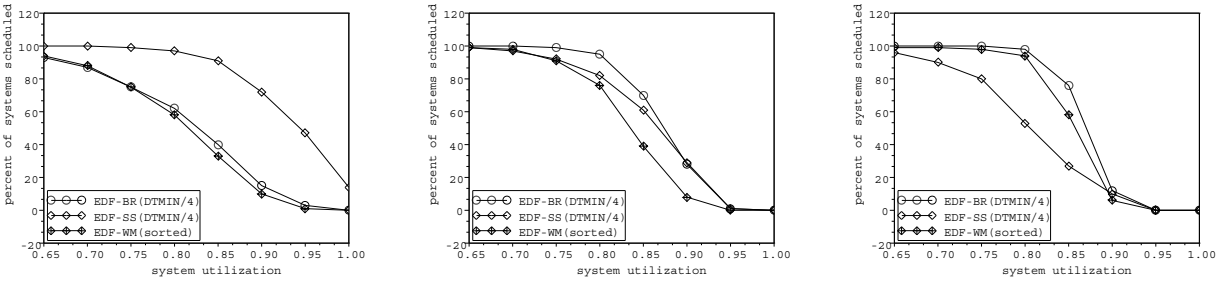
We have described a novel EDF-based real-time scheduling approach for multiprocessors, called EDF-BR. Tasks are statically allocated to processors but some of them are allowed to migrate between processors in a controlled manner. A processor bandwidth is used to control task migration so that schedulability is ensured. Costs due to migration are taken into consideration, an aspect usually neglected by other multiprocessor scheduling algorithms. The correctness of the proposed approach is shown and its performance is compared with related works. Results from experiments indicate that EDF-BR performs well. In particular, since the allocation procedure described here runs in polynomial time, EDF-BR is suitable for complex system running on many-processor architectures.

The results presented in this paper raise some interesting research questions. First, as the proposed bandwidth allocation is based on a given time slot T, heuristics to determine its value so that migration and preemptions are minimized are welcome. A possible approach is the clustering of tasks into subsets such that a value of T can be chosen for each



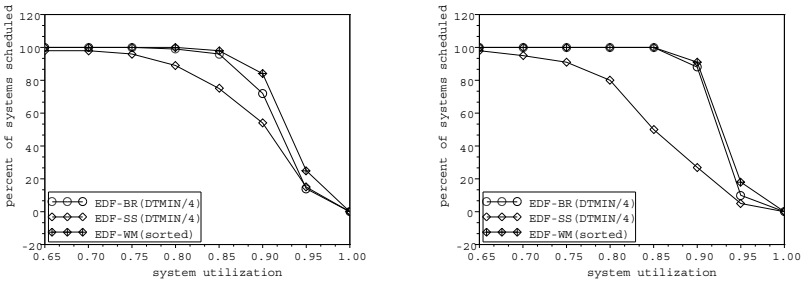
(a) (b) (c)

Figure 4. Tasks with constrained deadlines and utilization in [0.1;1.0]: (a) m=4; (b) m=8; (c) m=16.



(a) (b) (c)

Figure 5. Tasks with constrained deadlines and utilization in [0.5;1.0]: (a) m=4; (b) m=8; (c) m=16.



(a) (b)

Figure 6. Tasks with arbitrary deadlines and utilization in [0.5;1.0]: (a) m=8; (b) m=16.

cluster. Another issue is to consider aperiodic tasks. Since EDF-BR provides temporal isolation, it would be interesting to determine which values of T could be used to optimize their execution. These research topics will certainly be part of future works.

REFERENCES

[1] Björn Andersson and Konstantinos Bletsas. Sporadic multiprocessor scheduling with few preemptions. In *Proc. of the 20th IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, pages 243–252, 2008.

[2] Björn Andersson, Konstantinos Bletsas, and Sanjoy Baruah. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *Proc. of the 29th IEEE Real-Time Systems Symposium (RTSS)*, pages 385–394, 2008.

[3] Björn Andersson and Jan Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *Proc. of the 21th IEEE Real-Time Systems Symposium (RTSS) - Work-in-Progress Session*, pages 53–56, 2000.

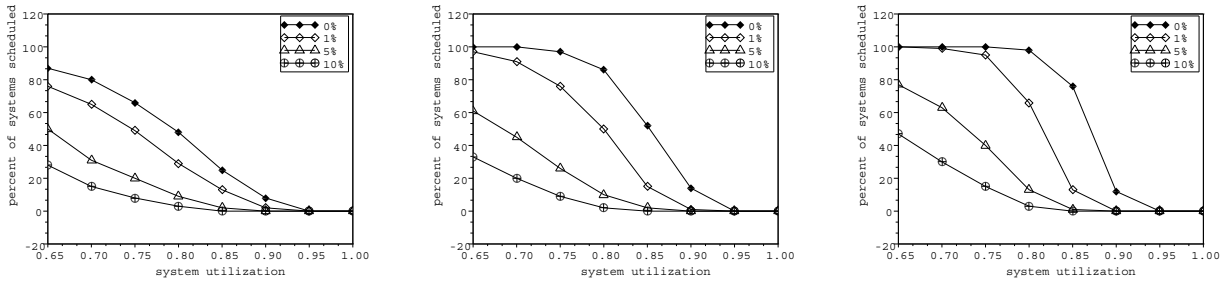
[4] Björn Andersson and Jan Jonsson. Preemptive multiprocessor scheduling anomalies. In *Proc. of the 16th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 12–19, 2002.

[5] Björn Andersson and Eduardo Tovar. Multiprocessor scheduling with few preemptions. In *Proc. of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 322–334, 2006.

[6] Theodore P. Baker. Multiprocessor edf and deadline monotonic schedulability analysis. In *Proc. of the 24th IEEE International Real-Time Systems Symposium (RTSS)*, page 120, 2003.

	$u \in [0.1; 1.0]$			$u \in [0.5; 1.0]$		
policy	m=4	m=8	m=16	m=4	m=8	m=16
EDF-BR(DTMIN/4)	0.00050	0.00113	0.00375	0.00038	0.00100	0.00284
EDF-SS(DTMIN/4)	0.15263	0.44375	0.51513	0.06438	0.22750	0.47350
EDF-WM(sorted)	0.15450	1.92167	10.33398	0.00250	0.03688	0.32615

Table I
MEAN TIME-DEMAND PER SYSTEM WITH CONSTRAINED DEADLINES (IN MINUTES).



(a) (b) (c)

Figure 7. Constrained deadlines, migration and utilization in [0.5;1.0]: (a) m=4; (b) m=8; (c) m=16.

[7] Theodore P. Baker, Michele Cirinei, and Marko Bertogna. Edzl scheduling analysis. *Real-Time Systems*, 40(3):264–289, 2008.

[8] Sanjoy Baruah and Theodore Baker. Schedulability analysis of global edf. *Real-Time Systems*, 38(3):223–235, 2008.

[9] Sanjoy Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[10] Sanjoy Baruah and Nathan W. Fisher. The partitioned dynamic-priority scheduling of sporadic task systems. *Real-Time Systems*, 36(3):199–226, 2007.

[11] Konstantinos Bletsas and Björn Andersson. Notional processors: An approach for multiprocessor scheduling. In *Proc. of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, 2009.

[12] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James H. Anderson, and Sanjoy Baruah. *A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms*. Chapman Hall/CRC, 2004.

[13] Hyeonjoong Cho, Binoy Ravindran, and E.Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proc. of the 27th IEEE Real-Time Systems Symposium (RTSS)*, pages 101–110, 2006.

[14] Nathan Fisher, Sanjoy Baruah, and Theodore P. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proc. of the 18th IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, pages 118–127, 2006.

[15] Rhan Ha and Jane W.S.Liu. Validating timing constraints in multiprocessor and distributed real-time systems. Technical report, University of Illinois at Urbana-Champaign, 1993.

[16] James H.Anderson, Vasile Bud, and UmaMaheswari C.Devi. An edf-based scheduling algorithm for multiprocessor soft real-time systems. In *Proc. of the 17th IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, pages 199–208, 2005.

[17] Jim Held, Jerry Bautista, and Sean Koehl. From a few cores to many: A tera-scale computing research review. Technical Report White paper, Intel Tera-scale Computing Research Program (Intel Corporation), 2006.

[18] Philip Holman and James H.Anderson. Adapting pfair scheduling for symmetric multiprocessors. *Journal of Embedded Computing*, 1(4):543–564, 2005.

[19] Shinpei Kato, Nobuyuki Yamasaki, and Yutaka Ishikawa. Semi-partitioned fixed-priority scheduling on multiprocessors. In *Proc. of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 23–32, 2009.

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

[21] R. Merritt. Cpu designers debate multi-core future. http://www.eetimes.com/showArticle.jhtml?articleID=206105179., June 2008.

[22] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Guillem Bernat, and Mateo Valero. Hardware support for wcet analysis of hard real-time multicore systems. In *Proc. of the 36th International Symposium on Computer Architecture (ISCA)*, pages 57–68, 2009.

[23] Xuefeng Piao, Sangchul Han, Heeheon Kim, Minkyu Park, Yookun Cho, and Seongje Cho. Predictability of earliest deadline zero laxity algorithm for multiprocessor real-time systems. In *Proc. of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 359–364, 2006.