Improved schedulability analysis of EDF on multiprocessor platforms

Marko Bertogna, Michele Cirinei and Giuseppe Lipari
Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: marko@sssup.it, cirinei@gandalf.sssup.it, lipari@sssup.it

Abstract

Multiprocessor hardware platforms are now being considered for embedded systems, due to their high computational power and little additional cost when compared to single processor systems. When scheduling real-time applications on multiprocessor platforms, a possibility is to use global scheduling*, where a scheduling algorithm dynamically assign tasks to processors, and tasks can* migrate *from one processor to another during their execution.*

In this paper, we tackle the problem of schedulability analysis of sporadic tasks in global scheduling systems, where the scheduler is the Earliest Deadline First (EDF) algorithm. We provide two main contributions. First, we show that two recently proposed tests perform poorly when the task set contains heavy tasks *(i.e. tasks with high utilization). We also show that neither test dominates the other. As a second contribution, we introduce a new schedulability test that improves significantly the percentage of accepted task sets, especially when considering task sets containing heavy tasks. We show the effectiveness of the proposed test through an extensive set of experiments.*

1. Introduction

Multiprocessor hardware platforms are becoming widespread and commonly used. Given the current limits of hardware technology, new increases in computational power can be obtained more easily and in a cost-effective way by using more than one processor rather than a more powerful single processor technology. As an example of this trend, the two major competitors in the high level microprocessor market, Intel and AMD, are both proposing dual core processors (i.e. chips with two processor cores) [2, 19].

Today, multiprocessors platforms are being used also in the embedded system domain. Example of such platforms include Philips Nexperia [20], Texas Instrument OMAP [15], Infineon Tricore [22].

Moreover, FPGA based solutions allow to customize the hardware structure of the system. The developer can choose the number of CPU cores, the buses and their connections to the memory and peripherals [1]. Some researchers have recently proposed hardware implementations of some parts of the operating system [17, 10, 6, 23, 16].

In contrast, a complete theory of real-time scheduling for multi-processor systems is still to come. We envision a future where a real-time application will be able to dynamically and adaptively scale from a single processor implementation to a multi-processor one, depending on the desired performance and level of quality of service.

In this paper we will address the problem of scheduling a set of real-time tasks in a symmetric multiprocessor (SMP) system consisting of m processors. This problem can be solved in two different ways: by partitioning tasks to processors, or with a global scheduler. In the first case, the tasks are allocated to processors at design time with an off-line procedure. The partitioning problem is analogous to the bin-packing problem, which is known to be NP-Hard in the strong sense [13]. However, once the tasks are allocated, the problem of scheduling is reduced to m single processor scheduling problems, for which optimal solutions are known. The main advantages of this approach are its simplicity and efficiency. Also, efficient communication and synchronization protocols have been proposed [11, 12, 21]. If the task set is fixed and known a-priori, partitioning approaches are probably the most appropriate solutions. On the other hand, if tasks can join and leave the system at run-time, it may be necessary to reconfigure the system by re-allocating tasks to processors.

We say that a task *migrates* if it is moved from one processor to another during its execution. A scheduling algorithm with migration assigns tasks to processors dynamically. Usually, such algorithms maintain a global shared queue for all processors, so they are referred to as *global schedulers*. The p-fair class of algorithms is known to be optimal for scheduling periodic real-time tasks with migration [8, 3]. Such algorithms are based on the concept of quantum: the time line is divided into equal-size intervals called quanta, and at each quantum the scheduler allocates tasks to processors. A disadvantage of this approach is that

Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)
0-7695-2400-1/05 $20.00 **IEEE**

all processors need to synchronize at the quantum boundary, when the scheduling decision is taken. Moreover, if the quantum is small, the overhead in terms of number of context switches and migrations may be too high. To avoid such overheads, some proposals are currently under investigation [3].

Recently, some researchers are addressing the problem of schedulability analysis of classical uniprocessor scheduling algorithms, like FP and EDF, on SMPs. Regarding schedulability analysis of periodic real-time tasks with EDF, Goossens Funk and Baruah [14] have recently proposed a schedulability test based on an utilization bound, assuming that tasks have relative deadlines equal to the period. Baker [7] proposed a different test extending the model to tasks with deadline less than or equal to period. Anderson et. al [5, 4] provided bounds to the utilization of feasible tasks sets scheduled with fixed priority.

The advantage of these schemes is the relatively simple implementation and the minor overhead in terms of number of context switches. However, many negative results are known for such schedulers. For example, EDF loses its optimality on multiprocessor platforms. Also, the overhead of migrating a task from one processor to another needs to be taken into account. In fact, in modern architectures, processors have a local cache memory, and migrating a task may invalidate the content of the cache.

Although global scheduling approaches seem to be complex or require too much overhead, we think that in some case they can be a valid option. For example, in some embedded processor architecture, with no cache and with simpler structures, the overhead of migration has a lower impact on the performance. Furthermore, in FPGA-based architectures, implementing the scheduler in HW can further reduce the overhead. From a theoretical point of view, we think that tackling the problem of global scheduling with EDF or FP algorithms can help understand the general problem of scheduling in multiprocessors.

1.1. Our contribution

This paper presents two main results. First, we discuss two recent solutions to the multiprocessor schedulability analysis using EDF, one proposed by Gooseens, Funk and Baruah [14], which will be denoted by GFB, and the other one proposed by Baker [7], which will be denoted by BAK. It was erroneously believed that the BAK test was more general than GFB, in that a task set schedulable by GFB was also schedulable by BAK. We prove that indeed this is not the case, and that neither test dominates the other.

Then, we propose an improved schedulability analysis that is able to successfully guarantee a larger portion of schedulable task set, especially in the case that *heavy tasks* (i.e. tasks whose utilization is greater than 0.5) are present. Finally, we show, with an extensive set of experiments, the improvements produced by our test.

The paper is organized as follows. In Section 2 we introduce the terminology and notation. In Section 3, the two main existing results on schedulability analysis with EDF are summarized and compared. In Section 4, we present our new test, which improves over the test proposed in [7]. In Section 5, the effectiveness of the proposed test is demonstrated through a set of experiments. Finally, in Section 6 we present our conclusions.

2. System model

We consider a set τ of periodic or sporadic tasks to be scheduled on m identical processors. Each task $\tau_k = (C_k, D_k, T_k) \in \tau$ is characterized by a worst-case computation time C_k, a period or minimum interarrival time T_k, and a relative deadline D_k. We denote with *implicit deadline* (resp. *constrained deadline*) the systems with $D_k \leq T_k$ (resp. $D_k = T_k$). We define the utilization of a task as $U_k = \frac{C_k}{T_k}$. We also define $\lambda_k = \frac{C_k}{D_k}$, which represents the "worst-case" request of a task in a generic time interval. Let $U_{\max}$ (resp. $\lambda_{\max}$) be the largest utilization (resp. the largest worst-case request) among all tasks: $U_{\max} = \max_{\tau_k \in \tau} \{U_k\}$ (resp. $\lambda_{\max} = \max_{\tau_k \in \tau} \{\lambda_k\}$).

A task is a sequence of jobs J_k^j, each job is characterized by an arrival time r_k^j and by a finishing time f_k^j. Goal of the scheduling algorithm is to guarantee that each job will complete before its absolute deadline $d_k^j = r_k^j + D_k$.

The *interference* over an interval $[a, b]$ on a task τ_k is the cumulative length of all interval in which the task is ready to execute but it cannot execute due to higher priority jobs. We will denote such interference with $I_k(a, b)$.

We also define the *interference* of a task τ_i on a task τ_k over an interval $[a, b]$ as $I_{i,k}(a, b)$ as the cumulative length of all intervals in which τ_k is ready to execute, τ_i is executing while τ_k is not. Notice that by definition:

$$I_{i,k}(a, b) \leq I_k(a, b), \quad \forall i, k, a, b. \tag{1}$$

The demand $df_k(a, b)$ of a task τ_k in an interval $[a, b]$ is the sum of the executions time of all jobs J_k^j with arrival time and deadline in $[a, b]$:

$$df_k(a, b) = \sum_{r_k^j \geq a \quad d_k^j \leq b} C_k.$$

Given a generic interval $[a, b]$, a job J_k^j can have arrival time before a and deadline in $[a, b]$. In such a case, the job may execute only a part of its computation time in $[a, b]$. The amount of execution time of such job in the interval $[a, b]$ is denoted as *carry-in* $\varepsilon_k(a, b)$ [7]. The workload

$W_k(a,b)$ of task τ_k in interval $[a,b]$ is the sum of the *carry-in* and of the demand:

$$W_k(a,b) = \varepsilon_k(a,b) + df_k(a,b).$$

The load $L_k(a,b)$ of a task τ_k in an interval $[a,b]$ is simply the workload divided by the length of the interval:

$$L_k(a,b) = \frac{W_k(a,b)}{b-a}.$$

To simplify the equations, we use $(x)_0$ as a short notation for $\max(0,x)$.

3. Summary of existing results

Recently, some authors tackled the problem of finding a schedulability test for periodic and sporadic real-time tasks on multiprocessor platforms, with a global EDF scheduling. We will describe here two different results that are both based on checking the utilization of the task set.

The first one, due to Goossens, Funk and Baruah [14] assumes an implicit deadline task model. It consists of a single simple inequality that compares the global utilization of the task set with a bound proved to be tight (in the sense that there is a task set with total utilization that exceeds by ϵ the bound, that EDF cannot schedule, $\forall \epsilon > 0$).

The second one, presented by Baker [7], is valid for constrained deadline task sets. The test consists in n conditions (one for each task) that must hold for the task set to be schedulable.

Now we briefly describe both results. Then we will compare them on specific examples and on a large number of randomly generated task sets.

3.1. The GFB test

According to the terminology introduced in [14], a *uniform* multiprocessor platform π consists of m equivalent processors, each one characterized by a *computing capacity* s_i. This means that a job that executes on the i-th processor for t time units completes $s_i \times t$ units of execution. Let S_π and s_π be the sum of the computing capacities of all processors and the computing capacity of the fastest processor of platform π, respectively. The first theorem was proved in [9].

Theorem 1 *A periodic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ is feasible on a uniform multiprocessor platform π having*

$$S_\pi = \sum_{\tau_i \in \tau} U_i$$

and

$$s_\pi = U_{max}$$

The following theorem shows a relation between an optimal algorithm for a uniform multiprocessor platform and EDF on a unit-capacity SMP.

Theorem 2 *A set of jobs I that is feasible on some uniform multiprocessor platform π with cumulative computing capacity S_π and in which the fastest processor has speed $s_\pi < 1$ is schedulable with EDF on the SMP π' composed by m processors with unit capacity, if*

$$m \geq \frac{S_\pi - s_\pi}{1 - s_\pi}.$$

Combining these results, the following schedulability test is obtained:

Theorem 3 (Goossens, Funk, Baruah) *A periodic task set τ is EDF-schedulable upon a SMP composed by m processors with unitary capacity, if*

$$\sum_{\tau_i \in \tau} U_i \leq m(1 - U_{max}) + U_{max}. \quad (2)$$

The previous test can be easily extended to the case of tasks with deadlines less than or equal to period. Note that Theorem 2 assumes an arbitrary collection of jobs. Theorem 1 can be easily modified as follows:

Lemma 1 *A task system τ composed by periodic and sporadic tasks is feasible on a uniform multiprocessor platform π which has*

$$S_\pi = \sum_{\tau_i \in \tau} \frac{C_i}{D_i} \quad (3)$$

and

$$s_\pi = \max_{\tau_i \in \tau} \left\{ \frac{C_i}{D_i} \right\}. \quad (4)$$

Proof.
An arbitrary task set τ with n tasks can always be scheduled on a uniform multiprocessor platform π composed by n processors, that for each task τ_i has a corresponding processor with computing capacity $s_i = C_i/D_i$. This can be done with an algorithm that allocates each task to the associated processor. The sum of the computing capacities of all processors and the computing capacity of the fastest processor of the platform π clearly respect conditions (3) and (4). □

By combining Lemma 1 with Theorem 2, it is possible to formulate a sufficient scheduling condition.

Theorem 4 (GFB) *A task set τ composed by both periodic and sporadic tasks is EDF-schedulable upon a SMP composed by m processors with unitary capacity, if*

$$\sum_{\tau_i \in \tau} \lambda_i \leq m(1-\lambda_{max}) + \lambda_{max}. \quad (5)$$

From now on we will refer with GFB to the test given by Equation (5).

3.2. The BAK test

Using a different approach, Baker derived a different sufficient schedulability condition [7]. The idea is based on the consideration that if a job J_k^j of task τ_k misses its deadline d_k^j, it means that the load in the interval $[r_k^j, d_k^j]$ is at least $m(1-\lambda_k)+\lambda_k$. The situation is depicted in Figure 1. Note that, to have a deadline miss for job J_k^j, all m processors have to execute other jobs for more than $D_k - C_k$. If it is possible to show, for every job J_k^j, that the task set cannot generate so much load in interval $[r_k^j, d_k^j]$, the schedulability is guaranteed.

Unfortunately, checking the condition directly in $[r_k^j, d_k^j]$ is not simple. Therefore, Baker proposes to *enlarge* the interval to find a better estimation of the carry-in of the interfering tasks. Instead of concentrating on interval $[r_k^j, d_k^j]$, Baker *extends* such interval in $[a, d_k^j]$. The basic idea is that $[a, d_k^j]$ is the largest possible interval such that the load is still greater than $m(1-\lambda_k)+\lambda_k$. This new interval is called *busy window*. By deriving an upper bound on the load produced in the busy window, the final result is obtained.

Theorem 5 (BAK) *A task set τ composed by n tasks is schedulable with EDF on a SMP with m processors if*

$$\forall \tau_k : \sum_{i=1}^{n} \min\{1, \beta_i\} \leq m(1-\lambda_k) + \lambda_k, \quad (6)$$

where

$$\beta_i = \begin{cases} U_i\left(1 + \frac{T_i - D_i}{D_k}\right) & \text{if} \quad \lambda_k \geq U_i \\ U_i\left(1 + \frac{T_i - D_i}{D_k}\right) + \frac{C_i - \lambda_k T_i}{D_k} & \text{if} \quad \lambda_k < U_i \end{cases}$$

3.3. Comparison between GFB and BAK

In his paper, Baker claims that test GFB is a special case of BAK, in the sense that task sets passing GFB are a subset of those passing BAK. To show this property, he introduced a simplified (less general) form of BAK. Unfortunately, due to an error in the derivation of the simplified form, the property is not true. We prove that there are schedulable task sets that pass GFB but not BAK, and vice versa.

The simplified test proposed by Baker says that a task set is EDF-schedulable on m processors if

$$\sum_{i=1}^{n} \min\left\{1, U_i\left(1 + \frac{T_i - D_i}{D_{min}}\right)\right\} \leq m(1-\lambda_{\max}) + \lambda_{\max} \quad (7)$$

where $D_{\min} = \min_{k=1}^{n}\{D_k\}$.

It is easy to see that for implicit deadline systems Equation (7) coincides with Equation (2). However, the simplified test (7) is not correctly derived from Equation (6). In fact, among the n inequalities of the general test, the minimum of the rightmost terms is chosen (the one with greatest λ_k), while the leftmost term is not necessarily the maximum one.

To highlight the problem, we present a counter-example. Consider a SMP with $m = 2$ processors and a task set τ consisting of 3 tasks:

$$\{\tau_1 = (49, 100, 100), \tau_2 = (49, 100, 100), \tau_3 = (2, 50, 100)\}.$$

with the notation $\tau_i = (C_i, D_i, T_i)$.

Applying Equation (6), the test for $k = 1$ and $k = 2$ is successful, but for $k = 3$ we have:

$$\frac{49}{100} + \frac{45}{50} + \frac{49}{100} + \frac{45}{50} + \frac{2}{100}\left(1 + \frac{50}{100}\right) < 2\left(1 - \frac{2}{50}\right) + \frac{2}{50}$$
$$\frac{281}{100} < \frac{196}{100}.$$

Therefore the task set does not pass the BAK test. However, it is easy to see that the task set passes the simplified test (7). Therefore, the simplified test is not a less general form of BAK, and it cannot be used to verify if BAK dominates GFB.

For implicit deadline systems, a correct simplification of Equation (6) is the following:

$$\forall \tau_k : \sum_{i=1}^{n} \min\{1, \beta_i\} \leq m(1-U_k) + U_k,$$

where

$$\beta_i = \begin{cases} U_i & \text{if} \quad U_k \geq U_i \\ U_i + \frac{C_i - U_k T_i}{T_k} & \text{if} \quad U_k < U_i \end{cases}$$

It consists of n inequalities, one of which coincides with Equation (2) of the GFB schedulability test. Therefore, it is possible to conclude that, for implicit deadline systems, Theorem 5 is a special case of Theorem 3.

However, for the general case of constrained deadline systems, it is not possible to conclude that one of the test is a special case of the other. In fact, by applying Equation (5) to the previous example we have:

$$\frac{49}{100} + \frac{49}{100} + \frac{2}{50} < 2\left(1 - \frac{49}{100}\right) + \frac{49}{100} \Leftrightarrow \frac{102}{100} < \frac{151}{100}$$

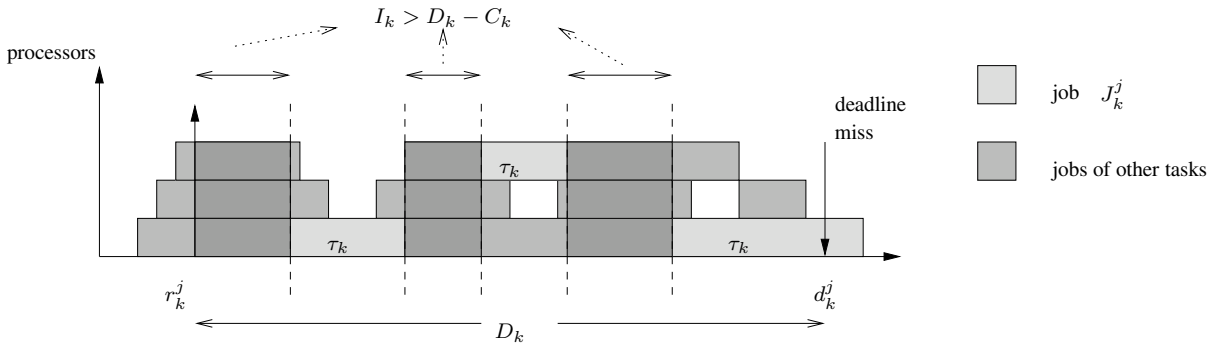

Figure 1. Problem window.

and the GFB test is passed, whereas BAK fails. It is also possible to show that there are task sets for which BAK gives a positive response and GFB does not.

3.4. Experimental evaluation of GFB and BAK

To evaluate the differences between the two tests and assess their effectiveness, we performed a set of experiments, measuring the percentage of task sets that are schedulable according to GFB and BAK with respect to the total percentage of schedulable task sets. The experiments were performed with RTSIM [18] tool[1].

We considered different SMP platforms. We performed a simulation for each combination of the following parameters: $m=3$, $m=5$, $m=10$ processors; $n=1.5\,m$, $n=3\,m$, $n=6\,m$ tasks; total utilization of the task sets varying from $0.2\,m$ to $0.8\,m$. To further consider the impact of tasks with high utilization factor, we classified the tasks into two categories: *heavy tasks* have utilization greater than 0.5 whereas *light tasks* have utilization less than 0.5. Experiments were made with a number of heavy tasks varying from 0 to 3.

Task sets were randomly generated according to the previous parameters. The period T_i of a task τ_i is taken from a set $\{100, 200, 300, \ldots, 1600\}$, each number having an equal probability of being selected. The deadline D_i follows a uniform distribution with a minimum value of $0.5\,T_i$ and a maximum value of T_i. The execution time C_i is accordingly computed from the generated utilization of the task (heavy or light).

We randomly generated 5,000 task sets for each experiment. For each generated task set we performed a simulation of the schedule up to the hyperperiod checking for missed deadlines, and we applied both GFB and BAK.

In Figure 2.a, we show the number of task sets passing each test (in relation to the total number of schedulable task sets) for the case of 9 tasks and 3 processors, with a total utilization $U_{\text{tot}} = 1.2$. One of the tasks is heavy, having utilization $U_{\text{max}} = 0.8$. Notice that the percentage of task sets for which both tests give a positive answer is low. Moreover, neither test dominates the other and there are many task sets for which BAK is successful whereas GFB is not, and vice versa.

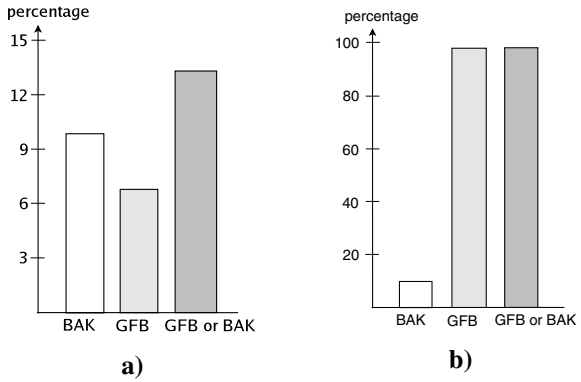

Figure 2. Percentage of schedulable task sets with GFB and BAK a) The case of 1 heavy task with utilization $U = 0.8$. b) The case of only light tasks.

If only light tasks are considered, GFB has a much better behavior, as shown in Figure 2.b. In this experiment we considered again 3 processors and 9 tasks, with total utilization equal to $U_{\text{tot}} = 1.2$, and all tasks with utilization less than 0.5. Notice that the percentage of schedulable task sets with BAK is again low, whereas GFB gives a positive response for almost all schedulable task sets.

It is worth to make some consideration on these simple experiments. Test GFB is very effective when analyzing task sets with a small U_{max}, as it is immediately evident from Equations (2) and (5). However, when U_{max} increases, the utilization bound becomes smaller and the test less ef-

[1] RTSIM is distributed under the Gnu Public License and it is available for download at http://rtsim.sssup.it/

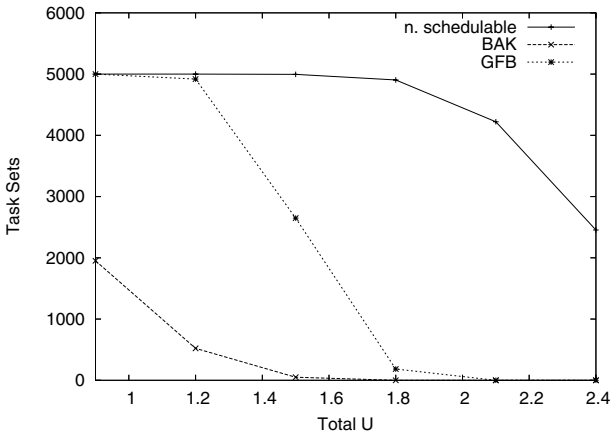

Figure 3. Experiments with 3 processors, 9 light tasks.

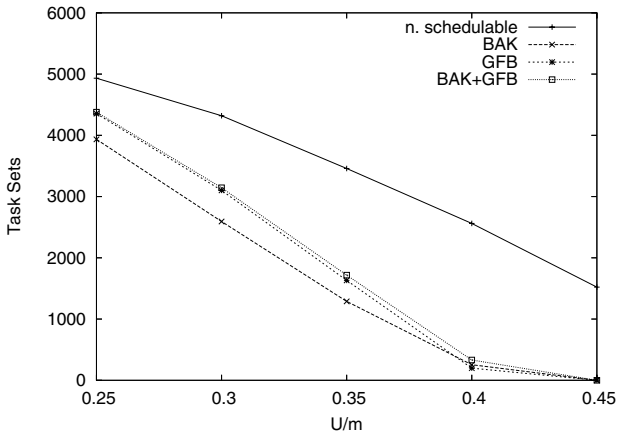

Figure 4. Experiments with 3 processors, one heavy task ($U_{\text{heavy}} = \frac{2}{3} U_{\text{tot}}$) and 9 light tasks.

fective. On the contrary, BAK does not depend on the value of $U_{\max}$ in a direct way, and it becomes more advantageous when considering task sets with one or more heavy tasks. As a general remark, for task sets including heavy tasks, none of the two tests is very effective, as there is a large subset of schedulable tasks for which GFB and BAK do not give a positive answer.

These conclusions are also supported by the plots shown in Figures 3 and 4.

In Figure 3, we considered 3 processors, 9 light tasks and a total utilization varying from $U_{\text{tot}} = 0.3\,m = 1.5$ to $U_{\text{tot}} = 0.8\,m = 4$. Notice that GFB gives a positive response in nearly all cases up to $U_{\text{tot}} = 1.2$, and then decreases abruptly, while BAK gives a positive response only in few cases.

In Figure 4, we considered 3 processors and 9 tasks. One of these tasks is heavy (utilization $U_{\max} = \frac{2}{3} U_{\text{tot}}$), while the others are light. In this case, BAK has performance very close to GFB, and there is some task set that is not schedulable according to GFB, for which BAK gives a positive response.

Starting from these considerations, a new test is presented in the next sections, which behaves better in presence of heavy tasks.

4. Scheduling analysis

In this section, we will extend the line of reasoning used in [7]. To clarify the methodology, we briefly describe the main steps that will be followed to derive the schedulability test.

1. As in [7], we start by assuming that a job J_k^j of task τ_k misses its deadline d_k^j;
2. Based on this assumption, we give a schedulability condition that uses the interference I_k that the job must suffer in interval $[r_k^j, d_k^j]$ for the deadline to be missed;
3. If we were able to precisely compute this interference in any interval, the schedulability test would simply consist in the condition derived at the precedent step and it would be necessary and sufficient; unfortunately, computing such interference is very difficult;
4. Therefore, we give an upper bound to the interference using the workload in the interval and derive a sufficient scheduling condition.

Let us first start by deriving some useful results on the interference that a task can suffer in an interval for the simultaneous execution of other tasks scheduled on the same multiprocessor platform.

4.1. Interference time

Lemma 2 *The interference that a task τ_i causes on a task τ_k in an interval $[a, b]$ is never greater than the workload of the task in the same interval:*

$$\forall i, k, a, b \quad I_{i,k}(a, b) \leq W_i(a, b) \leq b - a.$$

Lemma 2 is obvious, since $W_i(a, b)$ is an upper bound on the execution of τ_i in interval $[a, b]$.

Lemma 3 *The interference that a task τ_k can suffer in interval $[a, b]$ is the sum of the interferences of all other tasks*

(in the same interval) divided by the number of processors:

$$I_k(a,b) = \frac{\sum I_{i,k}(a,b)}{m}.$$

This descends from the observation that EDF is work-conserving, thus in the time instants in which a job is ready but not executing, each processor must be occupied by a job of another task.

Lemma 4 *$I_k(a,b) \geq x$ iff $\sum_i \min\left(I_{i,k}(a,b), x\right) \geq mx$.*

Proof.

Only If. Let ξ be the number of tasks for which it is $I_{i,k}(a,b) \geq x$.

$$\sum_{i=1}^{n} \min\left(I_{i,k}(a,b), x\right) = \xi x + \sum_{i:I_{i,k}<x} I_{i,k}(a,b) =$$
$$= \xi x + mI_k(a,b) - \sum_{i:I_{i,k}\geq x} I_{i,k}(a,b) \geq$$
$$\geq \xi x + mI_k(a,b) - \xi I_k(a,b) \geq \xi x + (m-\xi)x = mx.$$

If. Note that if $\sum_i \min\left(I_{i,k}(a,b), x\right) \geq mx$, it follows that

$$I_k(a,b) = \sum_i \frac{I_{i,k}(a,b)}{m} \geq \sum_i \frac{\min\left(I_{i,k}(a,b), x\right)}{m} \geq \frac{mx}{m} = x$$

□

Now we are ready to give the first schedulability condition. It is clear that, for a job to meet its deadline, the total interference on the task must be less than or equal to its slack time $D_k - C_k$. Hence, for a task to be schedulable, the condition must hold for all its jobs. We define the worst-case interference for task τ_k as:

$$\overline{I}_k = \max_j (I_k(r_k^j, d_k^j)) = I_k(r_k^{j*}, d_k^{j*}),$$

where $j*$ is the job instance in which the total interference is maximal. To simplify the notation, we define:

$$\overline{I}_{i,k} = I_{i,k}(r_k^{j*}, d_k^{j*})$$

Theorem 6 *The task set is schedulable iff, for each task τ_k one of the following is true:*

1) $\sum_{i\neq k} \min\left(\overline{I}_{i,k}, D_k - C_k\right) < m(D_k - C_k)$
2) $\sum_{i\neq k} \min\left(\overline{I}_{i,k}, D_k - C_k\right) = m(D_k - C_k)$ *and* $\exists h \neq k : 0 < \overline{I}_{h,k} \leq D_k - C_k$

Proof.

If. If 1) is true, from Lemma 4, we have that $\overline{I}_k < (D_k - C_k)$ and the task τ_k is feasible.

Now, suppose that 2) holds for task τ_k. It follows that task τ_h exists with $0 < \overline{I}_{h,k} \leq D_k - C_k$. There are less than m tasks with $\overline{I}_{i,k} > D_k - C_k$, otherwise it would be: $\sum_i \min\left(\overline{I}_{i,k}, D_k - C_k\right) > m(D_k - C_k)$, which contradicts the assumption.

In this situation, we substitute $D_k - C_k$ with $D_k - C_k + \epsilon$. The rightmost term increases by $m\epsilon$, while the leftmost term increases by less than $m\epsilon$ because less than m tasks will select the second term of the minimum:

$\forall \epsilon > 0 \ \sum_i \min\left(\overline{I}_{i,k}, D_k - C_k + \epsilon\right) < m(D_k - C_k + \epsilon)$, where ϵ is an arbitrarily small positive number. From Lemma 4: $\forall \epsilon > 0, \ \overline{I}_k < D_k - C_k + \epsilon$, which means that the interference that τ_k suffers is never greater than its slack time, so the task cannot have a deadline miss.

Only If. If it is: $\sum_i \min\left(\overline{I}_{i,k}, D_k - C_k\right) > m(D_k - C_k)$, then we have:
$\overline{I}_k = \frac{\sum_i \overline{I}_{i,k}}{m} \geq \frac{\sum_i \min\left(\overline{I}_{i,k}, D_k - C_k\right)}{m} > \frac{m(D_k - C_k)}{m} = D_k - C_k$, hence task τ_k is not schedulable.

Now suppose $\sum_i \min\left(\overline{I}_{i,k}, D_k - C_k\right) = m(D_k - C_k)$, but $\forall i : \overline{I}_{i,k} = 0$ *or* $\overline{I}_{i,k} > D_k - C_k$. There are exactly m tasks with $\overline{I}_{i,k} > D_k - C_k$. We have:
$\overline{I}_k = \frac{\sum_i \overline{I}_{i,k}}{m} > \frac{\sum_i \min\left(\overline{I}_{i,k}, D_k - C_k\right)}{m} = \frac{m(D_k - C_k)}{m} = D_k - C_k$, hence task τ_k is not schedulable. □

To better understand the key idea behind Theorem 6, consider again the situation depicted in Figure 1. It is clear that, if the interference that a task τ_i can impose on task τ_k in window $[r_k^j, d_k^j]$ is greater than $D_k - C_k$, it is sufficient to consider only the portion $D_k - C_k$ in the sum to verify the schedulability of task τ_k.

4.2. Workload

To apply the schedulability test of Theorem 6, the most straightforward approach is to compute, for each task τ_k, the interference $I_{i,k}(r_k^j, d_k^j)$ of each task τ_i, in every interval $[r_k^j, d_k^j]$, until the hyperperiod. This is actually impossible without a simulation of the system.

To avoid the complexity of this approach, we will use an upper bound on the interference. The test derived will then represent only a sufficient condition. From Lemma 2, we know that an upper bound on the interference $I_{i,k}(r_k^j, d_k^j)$ is the workload $W_i(r_k^j, d_k^j)$. The worst case for this workload is when the deadlines of job J_k^j and of the interfering task τ_i are aligned, because in this case the number of instances of τ_i that interfere with τ_k is maximum, as shown in [7].

Let $N_i = \left(\left\lfloor \frac{D_k - D_i}{T_i} \right\rfloor + 1\right)$ be the maximum number of jobs of τ_i that can be completely contained in $[r_k^j, d_k^j]$.

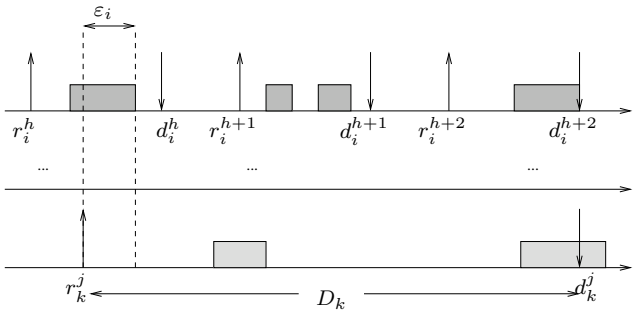

Figure 5. Carry in for task τ_i.

An upper bound on the workload of a task τ_k in a generic interval $[r_k^j, d_k^j]$ is:

$$W_i(r_k^j, d_k^j) \leq N_i C_i + \varepsilon_i(r_k^j, d_k^j)$$

where $\varepsilon_i(r_k^j, d_k^j)$ is the *carry-in* of task τ_i in interval $[r_k^j, d_k^j]$.

Lemma 5 *An upper bound on the carry-in is the following:*

$$\varepsilon_i(r_k^j, d_k^j) \leq \min\left(C_i, (D_k - N_i T_i)_0\right).$$

Proof.
Obviously, the carry-in of a task cannot exceed the worst-case computation time C_i of the task. Moreover, from Figure 5 it is easy to see that the carry-in cannot be greater than $D_k - N_i T_i$, when this term is positive. □

To further simplify the formula and to better appreciate the differences with the BAK test, we can express the schedulability test using the load. We denote with β_i an upper bound on the load of task τ_i in our interval of interest. In this case, for our interval $[r_k^j, d_k^j]$, we can write:

$$\beta_i = \frac{N_i C_i + \min\left(C_i, (D_k - N_i T_i)_0\right)}{D_k}.$$

Theorem 7 *The task set is feasible if, for each task τ_i one of the followings is true:*

$$\begin{array}{ll} 1) \sum_{i \neq k} \min\left(\beta_i, 1-\lambda_k\right) < m(1-\lambda_k) & \\ 2) \sum_{i \neq k} \min\left(\beta_i, 1-\lambda_k\right) = m(1-\lambda_k) \quad \textit{and} & (8) \\ \quad \exists i \neq k : \ 0 < \beta_i \leq 1-\lambda_k. & \end{array}$$

Proof.
Follows from Theorem 6. □

One of the main differences between our work and the results presented in [7] lies in term $(1 - \lambda_k)$ in the minimum. This term directly derives from term $D_k - C_k$ in Theorem 6. The underlying idea is that when considering the interference of a heavy task τ_i over τ_k, we do not want to overestimate its contribution to the total interference. If we consider its entire load, when we sum it together with the load of the other tasks on all m processors, its contribution could be much higher than $\frac{D_k - C_k}{D_k}$ and we could end up overestimating the total interference. Therefore, we must consider only the fraction of its workload that can actually interfere with task τ_k. This fraction is bounded by $1 - \lambda_k$.

As we show in the next section, this term will bring a substantial improvement on the schedulability test.

5. Experimental results

In this section, we compare our Equation (8) with GFB and BAK. To simplify the presentation, our test will be denoted by BCL. We used the same parameters already described in Section 3.4.

In Figure 6, we show the case of 5 processors, 2 heavy tasks with total utilization of 1.4 and 6 light tasks with total utilization varying from 0.1 to 1.1. The percentage of task sets that are schedulable by BCL is much higher than the ones obtained by GFB and BAK.

When we increase the number of processors, the results are similar. In Figure 7 we show the case of 10 processors, 2 heavy tasks with total utilization of 1.4 and 13 light tasks with total utilization varying from 0.1 to 1.9.

Increasing the number of heavy tasks, only BCL gives some positive result, whereas both GFB and BAK never give any response. An example of this situation is depicted in Figure 8, with 5 processors, 3 heavy tasks having total utilization 2.1, and only 5 light tasks with utilization varying from 0.1 to 0.5.

The good performances of BCL with one or more heavy tasks are due to term $1 - \lambda_k$ in Equation 8. For tasks with high utilization, this term is low, so that it will be selected in the minimum, bounding the overall interference.

Finally, in Figure 9 we show the results of an experiment consisting of 3 processors and 9 tasks, all light. In this case, test GFB is the best option, since it accepts almost all schedulable task sets for low utilization, whereas tests BAK and BCL behave badly. Without heavy tasks, the term $1 - \lambda_k$ of test BCL is high, and will never be selected in the minimum of Equation 8. In this case, BAK performs slightly better than BCL , since it can take advantage of its better estimation of the carry-in.

5.1. Considerations

From the experimental evaluation, GFB is the best test for task sets with small $U_{\max}$ (for example consisting only of light tasks). However, its performance degrades dramatically as $U_{\max}$ increases. On the other hand, the proposed test

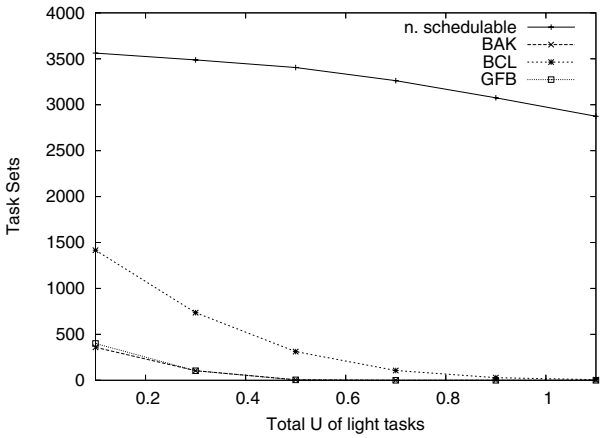

Figure 6. Experiment with 5 processors, 2 heavy tasks ($U_{\text{heavy tot}} = 1.4$) and 6 light tasks.

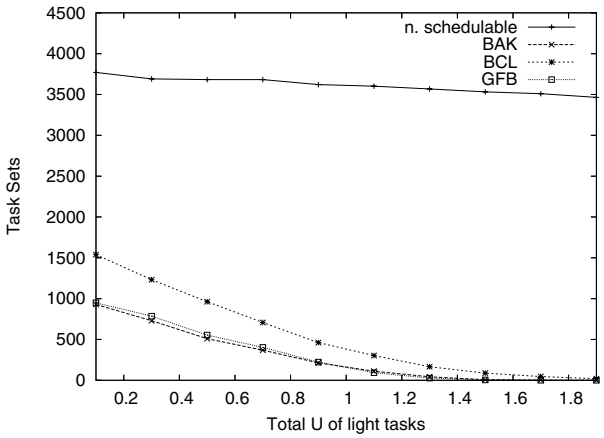

Figure 7. Experiment with 10 processors, 2 heavy tasks ($U_{\text{heavy tot}} = 1.4$) and 13 light tasks.

BCL behaves well when task sets with some heavy tasks are considered. In very rare cases, the BAK test may return a positive response where the other two fail.

All tests have a very low computational complexity: GFB has complexity equal to $O(n)$, whereas both BAK and BCL have complexity $O(n^2)$. Since GFB and BCL behave well in different situations, one possibility is then to run all three tests: if the task set is accepted by one of the tests, it can be guaranteed to be schedulable. The overall complexity of the resulting test is $O(n^2)$. Hence, for small task sets it is also possible to use this test for on-line admission control.

It is worth to make one final observation. In the general case, all proposed tests provide results that are far from the necessary and sufficient test (i.e. the simulation). We believe that there is still some space for improvement. We are currently investigating the possibility to extend our BCL test. One possibility is to improve the estimation of the carry-in, by extending the busy window with a technique similar to the one described by Baker in [7].

6. Conclusions

In this paper we tackled the problem of schedulability analysis of global scheduling systems, where the scheduling algorithm is EDF. We presented two main contributions. First, we have shown that, contrary to the current belief, the BAK test, proposed by Baker [7], does not dominate the GFB test, proposed by Gooseens, Funk and Baruah [14]. Moreover, both tests perform poorly when considering task sets with one or more heavy tasks. As second contribution, we proposed a novel schedulability test, BCL, that improves the percentage of accepted task sets when considering also tasks sets with heavy tasks. Given their low complexity, we propose to use a combination of the three tests. In fact, test GFB is more effective for tasks sets consisting only of light tasks, whereas BCL is more effective when considering heavy tasks.

However, as discussed in the previous section, we believe that there is still space for significant improvements. In fact, in many situations, the percentage of tasks sets accepted by any of the three tests with respect to the total number of schedulable task sets is still very low. We are currently investigating the possibility to extend our BCL tests to include more cases.

References

[1] Altera. http://www.altera.com/products/devices/stratix/stx-index.jsp. Web page, December 2004.

[2] AMD. http://www.amd.com/us-en/0,,3715_11787,00.html?redir=cpdc03. Web page, December 2004.

[3] J. Anderson and A. Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204, 2004.

[4] B. Andersson. *Static-priority scheduling on multiprocessors*. PhD thesis, Department of Computer Engineering, Chalmer University of Technology, Goteborg, Sweden, 2003.

[5] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In IEEE, editor, *Proceedings of the IEEE Real-Time Systems Symposium*, Dec 2001.

[6] N. Audsley and K. Bletsas. Fixed priority timing analysis of real-time systems with limited parallelism. In *Proceedings of the IEEE Euromicro Conference on Real Time Systems*, Catania, Italy, Jul 2004. IEEE.

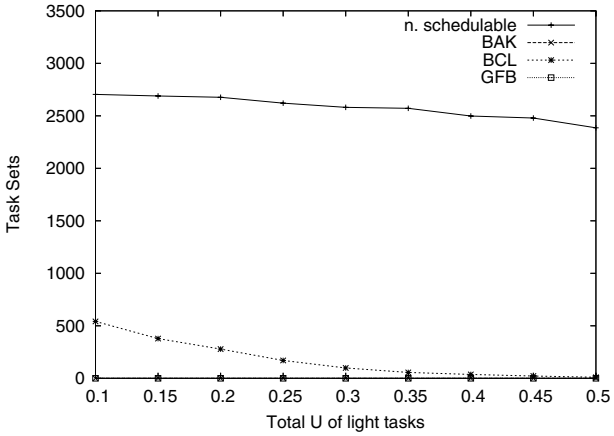

Figure 8. Experiment with 5 processors, 3 heavy tasks ($U_{\text{heavy tot}} = 2.1$) and 5 light tasks.

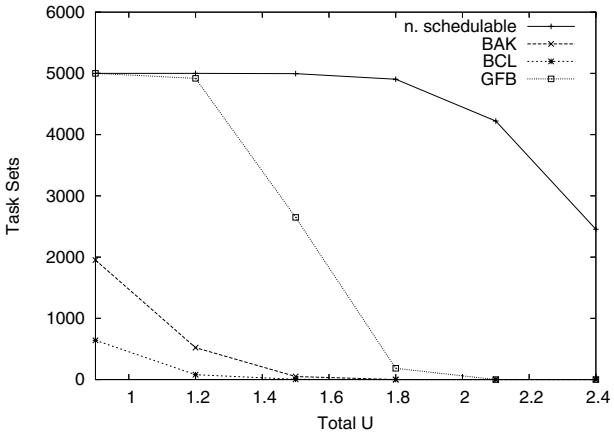

Figure 9. Experiment with 3 processors, 9 light tasks and no heavy task.

[7] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium, RTSS'03*, 2003.

[8] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 6, 1996.

[9] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In IEEE, editor, *Proceedings of the IEEE Real-Time Systems Symposium*, pages 183–192, Dec 2001.

[10] J. Furunas. Benchmarking of a real-time system that utilises a booster. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA2000*, Jun 2000.

[11] P. Gai, G. Lipari, and M. di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proceedings of the 22th Real-Time Systems Symposium*, December 2001.

[12] P. Gai, G. Lipari, and M. D. Natale. Design methodologies and tools for real-time embedded systems. *Special Issue of Design Automation for Embedded Systems*, 2002.

[13] M. Garey and D. Johnson. *Computers and Intractability*. W.H. Freman, New York, 1979.

[14] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, Sep-Oct 2003.

[15] T. Instruments. http://focus.ti.com/docs/prod/folders/print/omap5910.html. Web Page, December 2004.

[16] P. Kuacharoen, M. Shalan, and V. Mooney. A configurable hardware scheduler for real-time systems. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, pages 96–101, Jun 2003.

[17] L. Lindh, J. Starner, and J. Furunas. From single to multiprocessor real-time kernels in hardware. In *Proceedings of the IEEE Real Time Technology and Applications Symposium*, pages 15–17, Chicago, May 1995.

[18] L. Palopoli, G. Lipari, G. Lamastra, L. Abeni, G. Bolognini, and P. Ancilotti. An object oriented tool for simulating distributed real-time control systems. *Software - Practice and Experience*, 2002.

[19] PCWorld. http://www.pcworld.com/news/article/0,aid,117702,00.asp. Web page, December 2004.

[20] Philips. http://www.semiconductors.philips.com/products/nexperia/home/. Web page, December 2004.

[21] R. Rajkumar. Synchronization in multiple processor systems. In *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991.

[22] I. Technologies. http://www.infineon.com/cgi/ecrm.dll/ecrm/scripts/promotion_start_page.jsp?oid=52523. Web page, December 2004.

[23] M. Ward and N. Audsley. Hardware implementation of programming languages for real-time. In IEEE, editor, *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 276–285, 2002.