Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms

Marko Bertogna, Michele Cirinei, and Giuseppe Lipari, *Member*, *IEEE*

Abstract—This paper addresses the schedulability problem of periodic and sporadic real-time task sets with constrained deadlines preemptively scheduled on a multiprocessor platform composed by identical processors. We assume that a global work-conserving scheduler is used and *migration* from one processor to another is allowed during a task lifetime. First, a general method to derive schedulability conditions for multiprocessor real-time systems will be presented. The analysis will be applied to two typical scheduling algorithms: Earliest Deadline First (EDF) and Fixed Priority (FP). Then, the derived schedulability conditions will be tightened, refining the analysis with a simple and effective technique that significantly improves the percentage of accepted task sets. The effectiveness of the proposed test is shown through an extensive set of synthetic experiments.

Index Terms—Multiprocessor scheduling, real-time systems, global scheduling, task migration.

✦

---

1 INTRODUCTION

THE integration of multiple processors on a single chip constitutes one of the most important innovations in the design and development of modern embedded systems. In contrast, a complete theory of real-time scheduling for multiprocessor systems is still to come. Much of the research efforts in the past have been concentrated on scheduling and schedulability analysis of single-processor systems. Unfortunately, most of the results do not extend to multiprocessor systems.

In this paper, the problem of preemptively scheduling a real-time task set on a symmetric multiprocessor (SMP) system consisting of m processors is addressed. This problem can be solved in two different ways: by partitioning tasks to processors or with a global scheduler. In the first case, tasks are allocated to processors at design time with an offline procedure. The partitioning problem is analogous to the bin-packing problem, which is known to be NP-hard in the strong sense [1]. However, once the tasks are allocated, the scheduling problem is reduced to m single-processor scheduling problems, for which optimal solutions are known when preemptions are allowed. The main advantages of this approach are its simplicity and efficiency. If the task set is fixed and known a priori, in most cases, the partitioning approach is the most appropriate solution. On the other hand, if tasks can join and leave the system at runtime, it may be necessary to reconfigure the system by reallocating tasks to processors. As an example, consider a nonsaturated multiprocessor system, in which a task requests to join at a certain time and there is no processor with enough spare capacity to accommodate the new task. Thus, the partitioning algorithm

needs to be executed online to see if by reallocating some existing tasks, it is possible to accommodate the new one. Alternatively, a *load balancing* algorithm must be periodically executed to reallocate tasks to processors so as to avoid the potential waste of computational resources. The efficiency of the system depends on the frequency at which load-balancing routines are called and on the complexity of these algorithms. However, repeatedly calling nontrivial routines imposes a heavy load on the system, becoming infeasible for systems with highly variable workloads.

An alternative solution is represented by *global* schedulers, which maintain a single systemwide queue of ready tasks, from which tasks are extracted at runtime to be scheduled on the available computing resources. As opposed to partitioned approaches, different instances of the same task can execute on different processors. We say that a task *migrates* if it is moved from one processor to another during its lifetime. If tasks can change processors only at job boundaries, we say that *task migration* is allowed; we call instead *job migration* the possibility of moving a task from a processor to another during the execution of a job. Using global scheduling algorithms, tasks are dynamically assigned to the available processing units. This allows maintaining the system load to be always balanced, suggesting the use of global scheduling algorithms when the workload significantly varies at runtime or is not known a priori. An intermediate solution between global and partitioned scheduling is given by *semipartitioned* scheduling algorithms [2]. A semipartitioned scheduler limits the number of processors among which a task can migrate, simplifying the implementation of systems composed by a large number of processors and reducing the penalties associated to task migrations.

The Pfair class of global scheduling algorithms is known to be optimal for scheduling periodic and sporadic real-time tasks with job migration when deadlines are equal to periods [3], [4]. Such algorithms are based on the concept of quantum (or slot): the time line is divided into equal-size intervals called quanta, and at each quantum, the scheduler

---

● *The authors are with the Scuola Superiore Sant'Anna, piazza Martiri della Libertà 33, 56127 Pisa, Italy.*
E-mail: {m.bertogna, m.cirinei, lipari}@sssup.it.

---

allocates tasks to processors. A disadvantage of this approach is that 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 the number of context switches and migrations may be too high. Solutions to mitigate this problem have been proposed in the literature [5]; however, the complexity of their implementation increases significantly.

A more reasonable number of context switches can be obtained by reducing the number of times the priority of a job can change. Using a *task-level fixed-priority* (FP) scheduler, all jobs generated by the same task have identical priorities. A *job-level FP* scheduler, instead, can change the priority of a task only at job boundaries. An example of such a scheduler is given by Earliest Deadline First (EDF). Note that Pfair algorithms can change the priority of a job even during its execution. Such kind of schedulers are called *job-level dynamic.* The advantage of using a task- or job-level FP scheduler is the relatively simple implementation and the minor overhead. However, the overhead of migrating a task from one processor to another still needs to be taken into account.

The schedulability analysis of job and task-level FP scheduling algorithms on SMPs has only recently been addressed [6], [7], [8], [9], [10], [11], [12], [13], [14]. The feasibility problem appears to be much more difficult than in the uniprocessor case. For example, EDF loses its optimality on multiprocessor platforms. Due to the complexity of the problem, only sufficient conditions have been derived so far. As shown in our simulations, the existing schedulability tests consider situations that are overly pessimistic, leading to a significant number of rejected task sets that are instead schedulable.

1.1 Our Contribution

This paper presents nontrivial improvements on schedulability analysis of global scheduling algorithms for multiprocessor systems. The contributions of our analysis are manifold. First, general conditions that are valid for any work-conserving scheduling algorithm and constrained deadline task sets are stated. They are later adapted to two popular scheduling algorithms: FP and EDF. These tests can successfully guarantee a larger portion of schedulable task sets when *heavy tasks* (i.e., tasks whose utilization is greater than 0.5) are present.

Second, the main weak points of these tests are identified. These observations trigger a further refinement on the computation of the interference a task can be subject to. The result of this latter step is a novel iterative algorithm that allows considerably increasing the number of successfully detected schedulable task sets, compared to *any* previously proposed schedulability test. The complexity of the proposed algorithm is pseudopolynomial but can be reduced by limiting the number of iterations of the test to a small constant, without significantly affecting performances. Finally, an extensive set of synthetic experiments is presented to show the improved performances of our analysis.

2 System Model

Consider a set τ composed by n periodic or sporadic tasks to be preemptively scheduled on m identical processors, using a global scheduler with job migration support.

A task τ_k is a sequence of jobs J_k^j, where each job is characterized by an arrival time r_k^j, an absolute deadline d_k^j, a computation time c_k^j, and a finishing time f_k^j. Every 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, with $C_k \geq c_k^j$, $r_k^j \geq r_k^{(j-1)} + T_k$, and $d_k^j = r_k^j + D_k$. We denote with *constrained deadline* (respectively, *implicit deadline*), the systems with $D_k \leq T_k$ (respectively, $D_k = T_k$). *This paper will exclusively consider implicit and constrained deadline systems*, leaving the analysis of arbitrary deadlines as a future work.

We define the utilization of a task as $U_k = \frac{C_k}{T_k}$. We also define the density $\lambda_k = \frac{C_k}{D_k}$, which represents the "worst case" request of a task in a generic time interval. Let $U_{\max}$ (respectively, $\lambda_{\max}$) be the largest utilization (respectively, the largest density) among all tasks. The total utilization U_{tot} and the total density λ_{tot} of a task set are defined as $U_{tot} = \sum_{\tau_k \in \tau} U_k$ and $\lambda_{tot} = \sum_{\tau_k \in \tau} \lambda_k$. To simplify the equations, we use $(x)_0$ as a short notation for $\max(0, x)$.

We assume that the cost of preemption and migration are either negligible or included in the worst case execution parameters. Moreover, job parallelism is forbidden, meaning that no job of any task can be executed at the same time on more than one processor. Unless otherwise stated, we make no assumption on the global scheduling algorithm in use, except that it should be *work conserving*, according to the following definition.

Definition 1 (work conserving). *A scheduling algorithm is work conserving if there are no idle processors when a ready task is waiting for execution.*

2.1 Workload and Interference

The workload $W_k(a, b)$ of a task τ_k in an interval $[a, b]$ is the amount of time task τ_k executes during interval $[a, b)$, according to a given scheduling policy. The *interference* over an interval $[a, b)$ on a task τ_k is the cumulative length of all intervals in which τ_k is ready to execute but it cannot execute due to higher priority jobs. We denote such interference with $I_k(a, b)$. We also define the *interference* $I_{i,k}(a, b)$ of a task τ_i on a task τ_k over an interval $[a, b)$ as the cumulative length of all intervals in which τ_k is ready to execute and τ_i is executing while τ_k is not. Notice that by definition

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

2.2 Time Division

Despite the fact that for mathematical convenience, time instants and interval lengths are often modeled using real numbers, in a real implemented system, time is not infinitely divisible. The times of event occurrences and the durations between them cannot be determined more precisely than one tick of the system clock. Therefore, any time value t involved in scheduling is assumed to be a nonnegative integer value and is viewed as representing the

entire interval $[t, t+1)$. This convention allows the use of mathematical induction on clock ticks for proofs, avoids potential confusion around end points, and prevents impractical schedulability results that rely on being able to slice time at arbitrary points.

3 SUMMARY OF EXISTING RESULTS

To our knowledge, this is the first work explicitly deriving schedulability conditions that are valid in general for any global scheduling algorithm. However, there are results on the schedulability analysis of systems scheduled with a particular policy, like EDF or FP.

Regarding the schedulability analysis of periodic real-time tasks with EDF, Goossens et al. [6] proposed a schedulability test based on a utilization bound, assuming that tasks have relative deadlines equal to the period. It consists of a single simple inequality that compares the global utilization of the task set with a bound proven to be tight (in the sense that there are task sets with total utilization exceeding the bound by ϵ, which EDF cannot schedule, $\forall \epsilon > 0$).

Theorem 1 (from [6]). *A task set τ composed by periodic and sporadic tasks with implicit deadlines is EDF-schedulable upon an SMP composed by m processors with unitary capacity if*

$$U_{tot} \leq m(1 - U_{\max}) + U_{\max}. \qquad (2)$$

We will hereafter show how to modify the above result when deadlines can be different than periods.

According to the terminology introduced in [6], 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 ith 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 following theorem shows a relation between an optimal algorithm for a uniform multiprocessor platform and EDF on a unit-capacity SMP.

Theorem 2 (from [6]). *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 an SMP π' composed by m processors with unit capacity if*

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

Note that Theorem 2 assumes an arbitrary collection of jobs. The next lemma states a feasibility result that instead applies to periodic and sporadic task sets.

Lemma 1. *A task system τ composed by periodic and sporadic tasks with constrained deadlines is feasible on a uniform multiprocessor platform π that has $S_\pi = \lambda_{tot}$ and $s_\pi = \lambda_{\max}$.*

Proof. An arbitrary task set τ with n tasks can always be scheduled on a uniform multiprocessor platform π composed by n processors, which 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 are therefore equal to, respectively, λ_{tot} and $\lambda_{\max}$.   □

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

Theorem 3 (GFB). *A task set τ composed by both periodic and sporadic tasks with constrained deadlines is EDF-schedulable upon an SMP composed by m processors with unitary capacity if*

$$\lambda_{tot} \leq m(1 - \lambda_{\max}) + \lambda_{\max}. \qquad (3)$$

When deadlines are equal to periods, the above result reduces to the utilization-based schedulability condition of Theorem 1. From now on, we will refer with GFB the test given by (3).

A drawback of the GFB test is that it cannot guarantee the schedulability of task systems having at least one task with large execution requirements: When $\lambda_{\max}$ is large, the right-hand side of (3) remains small even when increasing the number of processors. This is due to a particular effect, called Dhall's effect [15], that limits the total schedulable utilization of systems scheduled with EDF (or FP). Since GFB makes use of a very small number of parameters, it is not able to distinguish whether this effect can take place. To overcome this limit, Goossens et al. proposed in [6] a modified version of EDF, called EDF^k, assigning the highest priority to the k heaviest tasks and scheduling the remaining ones with EDF. A schedulability test is found by applying GFB to a subsystem composed by the $(n-k)$ EDF-scheduled tasks on $(m-k)$ processors.[1] To increase the chances of finding a feasible schedule, it is then possible to try all possible values for $k \in [0, m)$. Another possibility to overcome Dhall's effect is to assign the highest priority to tasks having a utilization larger than a given threshold, as proposed in [17]. This algorithm, called EDF with Utilization Separation (EDF-US), allows reaching a (tight) schedulable utilization bound of $\frac{m+1}{2}$ for implicit deadline systems, when a threshold of $\frac{1}{2}$ is used [18]. A generalization of the above results for systems with deadlines different from periods is presented in [16].

A different analysis for constrained deadline systems scheduled with global EDF or FP has been proposed by Baker in [7]. A test is derived, consisting of n conditions (one for each task) that must hold for the task set to be schedulable. The idea is based on the consideration that if a job J_k^j of a task τ_k misses its deadline d_k^j, it means that the load in an interval $[r_k^j, d_k^j)$, called the *problem window*, is at least $m(1 - \lambda_k) + \lambda_k$. The situation is depicted in Fig. 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.

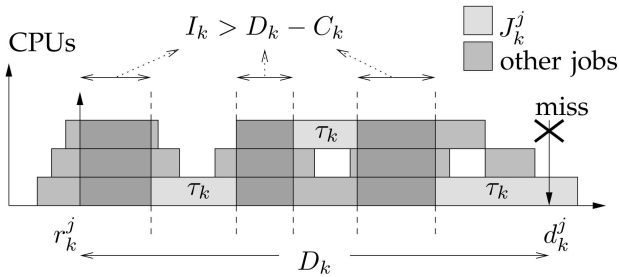1. The proof for systems with deadlines different from periods can be found in [16].

Fig. 1. Problem window.

The interference of any task τ_i on task τ_k in interval $[r_k^j, d_k^j)$ may include one job of τ_i with arrival before r_k^j and deadline in $[r_k^j, d_k^j)$ that execute entirely or in part inside the interval. The contribution of this job to the interference is called *carry-in* (it will be defined more precisely in Section 4.2).

To find a better estimation of the carry-in of the interfering tasks, Baker proposes to *enlarge* the considered interval: Instead of concentrating on interval $[r_k^j, d_k^j)$, he *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 the *busy window*. By deriving an upper bound on the load produced in the busy window, a sufficient schedulability condition is obtained [7, Theorem 12].

Following a similar approach, Baker later refined his analysis for EDF [13], [14] and for FP [12], [14], considering also the case in which deadlines can be greater than periods. The test in [13] is proved to generalize the utilization bound of Goossens et al. [6] for implicit deadline systems. However, the dominance relation ceases when considering deadlines different from periods, as we will show in our simulations.

Among previous works addressing the schedulability analysis of systems scheduled with FP, Andersson et al. [8], [9] provided bounds to the schedulable utilization of tasks sets scheduled using rate-monotonic priority assignment. They proved that an implicit deadline task set can be successfully scheduled on m processors if the total utilization is at most $m^2/(3m - 2)$ and every task has an individual utilization less than or equal to $m/(3m - 2)$. Using this result, they showed that an algorithm called RM-US$[m/(3m - 2)]$—which gives the highest priority to the tasks with a utilization greater than $m/(3m - 2)$ and schedules the other ones with rate monotonic—is able to reach a schedulable utilization of $m^2/(3m - 2)$. These bounds have been later improved in [11], where the following density-based test is derived.

Theorem 4 (from [11]). *A set of periodic or sporadic tasks with constrained deadlines is schedulable with Deadline-Monotonic (DM) priority assignment on $m \geq 2$ processors if $\lambda_{tot} \leq \frac{m}{2}(1 - \lambda_{max}) + \lambda_{max}$.*

When deadlines are equal to periods, the above condition is shown to dominate the rate-monotonic result in [8]. A corollary of Theorem 4 is that using a hybrid version of deadline monotonic—called DM-DS[1/3]—that gives the highest priority to tasks with a density higher than 1/3, it is

possible to schedule every constrained deadline task set with $\lambda_{tot} \leq (m + 1)/3$.

When considering dynamic-job priority scheduling algorithms, there are recently proposed solutions [20] that have good schedulability performances, with a number of context changes lower than that with Pfair algorithms. An interesting algorithm that has the same worst case number of preemptions as EDF but much better scheduling performances for multiprocessor systems is EDF with zero laxity (EDZL). The schedulability conditions for EDZL have been derived in [21] and [22].

4 SCHEDULING ANALYSIS

In this section, we will extend the line of reasoning used in [7], [13], [12], and [14]. 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 of the condition derived at the preceding step, and it would be necessary and sufficient; unfortunately, we are not able to find a method to exactly compute such interference with reasonable complexity.
4. Therefore, we give an upper bound to the interference in the interval and derive a sufficient scheduling condition.

Let us first start by deriving some useful results on the interference time.

4.1 Interference Time

The results contained in this section apply to any collection of tasks scheduled with a work-conserving policy. No other assumption is made on the scheduling algorithm in use.

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. *For a work-conserving scheduler, the following relation holds:*

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

Since the scheduling algorithm in use is work conserving, in the time instants in which a job is ready but not executing, each processor must be occupied by a job of another task. Since $I_{k,k}(a, b) = 0$, we can exclude the contribution of τ_k to the interference.

Lemma 4.

$$I_k(a,b) \geq x \iff \sum_{i \neq k} \min\big(I_{i,k}(a,b), x\big) \geq mx.$$

Proof. *Only If.* Let ξ be the number of tasks for which $I_{i,k}(a,b) \geq x$. If $\xi > m$, then $\sum_{i \neq k} \min(I_{i,k}(a,b), x) \geq \xi x > mx$. Otherwise, $(m - \xi) \geq 0$, and using Lemma 3 and (1),

$$\begin{aligned}
\sum_{i \neq k} \min\big(I_{i,k}(a,b), x\big) &= \xi x + \sum_{i: I_{i,k} < x} I_{i,k}(a,b) \\
&= \xi x + m I_k(a,b) - \sum_{i: I_{i,k} \geq x} I_{i,k}(a,b) \\
&\geq \xi x + m I_k(a,b) - \xi I_k(a,b) \\
&= \xi x + (m - \xi) I_k(a,b) \geq \xi x + (m - \xi) x \\
&= mx.
\end{aligned}$$

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

$$I_k(a,b) = \sum_{i \neq k} \frac{I_{i,k}(a,b)}{m} \geq \sum_{i \neq k} \frac{\min\big(I_{i,k}(a,b), x\big)}{m} \geq \frac{mx}{m} = x.$$

□

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 in the interval between the release time and the deadline of the job 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

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

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

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

Theorem 5. *A task set τ is schedulable on a multiprocessor composed by m identical processors iff for each task τ_k*

$$\sum_{I \neq k} \min\big(\overline{I}_{i,k}, D_k - C_k + 1\big) < m(D_k - C_k + 1). \qquad (4)$$

Proof. *If.* If (4) is valid, from Lemma 4, we have $\overline{I}_k < (D_k - C_k + 1)$. Therefore, for the integer-time assumptions, job J_k^{j*} will be interfered for at most $D_k - C_k$ time units. From the definition of interference, it follows that J_k^{j*} (and, therefore, every other job of τ_k) will complete after at most D_k time-units, and the task τ_k is schedulable.

Only If. The proof is by contradiction. If $\sum_{i \neq k} \min(\overline{I}_{i,k}, D_k - C_k + 1) \geq m(D_k - C_k + 1)$, then $\overline{I}_k = \frac{\sum_{i \neq k} \overline{I}_{i,k}}{m} \geq \frac{\sum_{i \neq k} \min(\overline{I}_{i,k}, D_k - C_k + 1)}{m} \geq \frac{m(D_k - C_k + 1)}{m} = D_k - C_k + 1$, hence, task τ_k is not schedulable. □
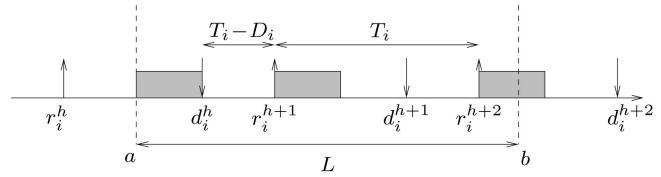

Fig. 2. Densest possible packing of jobs of task τ_i in interval $[a,b]$.

To better understand the key idea behind Theorem 5, consider again the situation depicted in Fig. 1. It is clear that when a task τ_k is executing, it cannot be interfered. To check the schedulability of τ_k, we do not want to consider as interfering contribution the work done in parallel by other tasks while τ_k is executing. If τ_k does not miss its deadline, it will execute for C_k time units, and the total interference is strictly less than $(D_k - C_k + 1)$. The theorem says that to check if τ_k can suffer enough interference in a window $[r_k^j, d_k^j)$ to cause a deadline miss, it is sufficient to consider the sum of the interfering contributions $I_{i,k}(r_k^{j*}, d_k^{j*})$ of the other tasks τ_i, limiting each contribution to at most $(D_k - C_k + 1)$ time units.

4.2 Workload

The necessary and sufficient schedulability condition expressed by Theorem 5 cannot be used to check if a task set is schedulable without knowing how to compute the interference terms $\overline{I}_{i,k}$'s. Unfortunately, we are not aware of any strategy that can be used to compute the worst case interferences starting from the given task parameters. To sidestep this problem, 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 $\overline{I}_{i,k}$ is the workload $W_i(r_k^{j*}, d_k^{j*})$. Since evaluating the worst case workload is still a complex task, we will again use an upper bound on it.

To derive a safe upper bound on the workload that a task can produce in a considered interval, we are interested in finding the densest possible packing of jobs that can be generated by a legal schedule. Since at this moment, we are not relying on any particular scheduling policy, no information can be used on the priority relations among the jobs involved in the schedule.

To simplify the presentation, we will call *carry-in* ε_k of a task τ_k in an interval $[a,b)$ the amount of execution produced by a job of τ_k having a release time before a and a deadline after a. Similarly, the *carry-out* z_k will be the amount of execution of a job of τ_k having a release time in $[a,b)$ and a deadline after b. Notice that in the constrained deadline model, there is at most one carry-in and one carry-out job. We will denote them, respectively, as J_i^ε and J_i^z.

With these premises and as long as there are no deadline misses, a bound on the workload of a task τ_i in a generic interval $[a,b)$ can be computed considering a situation in which the carried-in job J_i^ε starts executing as close as possible to its deadline and right at the beginning of the interval (therefore, $a = d_i^\varepsilon - C_i$) and every other instance of τ_i is executed as soon as possible. The situation is represented in Fig. 2.

Since a job J_i^j can be executed only in $[r_i^j, d_i^j)$ and for at most C_i time units, it is immediate to see that the depicted

situation provides the highest possible amount of execution in interval $[a, b)$: moving the interval backwards, the carry-in cannot increase, while the carry-out can only decrease. Instead, advancing the interval, the carry-in will decrease, while the carry-out can increase by at most the same amount. The situation is periodic.

Based on Fig. 2, we now compute the effective workload of task τ_i in an interval $[a.b)$ of length L in the situation described above. Note that the first job of τ_i after the carry-in is released at time $a + C_i + T_i - D_i$. The next jobs are then released periodically every T_i time units. Therefore the number $N_i(L)$ of jobs of τ_i that contribute with an entire WCET to the workload in an interval of length L is at most $\left(\left\lfloor \frac{L - (C_i + T_i - D_i)}{T_i} \right\rfloor + 1 \right)$. Therefore,

$$ N_i(L) = \left\lfloor \frac{L + D_i - C_i}{T_i} \right\rfloor. \tag{5} $$

The contribution of the carried-out job can then be bounded by $\min(C_i, L + D_i - C_i - N_i(L)T_i)$. A bound on the workload of a task τ_i in a generic interval of length L is then

$$ \mathcal{W}_i(L) = N_i(L)C_i + \min(C_i, L + D_i - C_i - N_i(L)T_i). \tag{6} $$

We are now ready to state the first polynomial complexity schedulability test valid for task systems scheduled with work-conserving global scheduling policies on multiprocessor platforms.

Theorem 6. *A task set τ is schedulable with any work-conserving global scheduling policy on a multiprocessor platform composed by m identical processors if for each task $\tau_k \in \tau$*

$$ \sum_{i \neq k} \min(\mathcal{W}_i(D_k), D_k - C_k + 1) < m(D_k - C_k + 1). \tag{7} $$

Proof. Since no assumption has been made on the scheduling algorithm used, (6) is valid for any work-conserving scheduling algorithm. Using Lemma 2, we then have

$$ \overline{I}_{i,k} = I_{i,k}\left(r_k^{j*}, r_k^{j*} + D_k\right) \leq W_i\left(r_k^{j*}, r_k^{j*} + D_k\right) \leq \mathcal{W}_i(D_k). $$

The theorem follows from Theorem 5, using $\mathcal{W}_i(D_k)$ as an upper bound for $\overline{I}_{i,k}$.   □

The above schedulability test consists of n inequalities and can be performed in polynomial time.

Nevertheless, when the algorithm in use is known, this information can be used to derive tighter conditions. As an example, we will hereafter consider the EDF and FP cases.

4.3 Schedulability Test for EDF
When tasks are scheduled according to EDF, a better upper bound can be found for $\overline{I}_{i,k}$. The worst case situation can be improved by noting that no carried-out job can interfere with task τ_k in the considered interval $[r_k^{j*}, d_k^{j*})$: Since a carry-out job has, by definition, a deadline after d_k^{j*}, it will have a lower priority than τ_k, according to EDF. We can then refine the worst case situation to be used to compute an upper bound on the interference of a task in $[r_k^{j*}, d_k^{j*})$. As shown in [7], we can consider the situation in which the
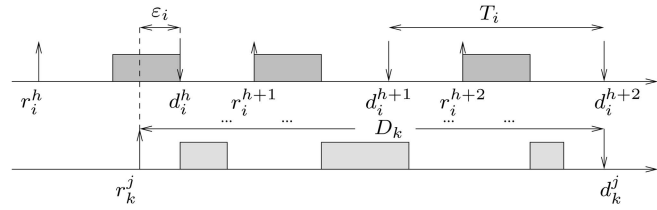

Fig. 3. Scenario that produces the maximum possible interference of task τ_i on a job of task τ_k when EDF is used.

carried-out job J_i^z has its deadline at the end of the interval—i.e., coincident with a deadline of τ_k—and every other instance of τ_i is executed as late as possible. The situation is depicted in Fig. 3.

An upper bound on the interference can then be easily derived by analyzing the above situation. We will again consider the workload in the corresponding interval $[r_k^{j*}, d_k^{j*})$ of length D_k. There are many possible formulas to express such workload. We choose to separate the contribution of the first job contained in the interval (not necessarily the carry-in job) from the rest of the jobs of τ_i. Each one of the jobs after the first one contributes for an entire worst-case computation time. There are $\left\lfloor \frac{D_k}{T_i} \right\rfloor$ such jobs. Instead, the first job contributes for $D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i$, when this term is lower than C_i. We therefore obtain the following expression:

$$ \overline{I}_{i,k} \leq \left\lfloor \frac{D_k}{T_i} \right\rfloor C_i + \min\left(C_i, D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i \right) \doteq \mathcal{I}_{i,k}. \tag{8} $$

A schedulability test for EDF immediately follows.

Theorem 7. *A task set τ is schedulable with global EDF on a multiprocessor platform composed by m identical processors if for each task $\tau_k \in \tau$*

$$ \sum_{i \neq k} \min\left(\mathcal{I}_{i,k}, D_k - C_k + 1 \right) < m(D_k - C_k + 1). \tag{9} $$

For EDF-scheduled systems, this condition is tighter than the condition expressed by Theorem 6.

4.4 Schedulability Test for FP
When analyzing a task set scheduled with FP, the upper bound on the interference given by (8) cannot be used. Nevertheless, it is still possible to use the general bound given by (6) that is valid for any work-conserving scheduling policy. However, the tightness of this bound can be significantly improved for FP by noting that the interference from tasks with a lower priority is always null. Theorem 6 can then be modified by limiting the sum of the interfering terms to the tasks with a priority higher than τ_k's. The following theorem assumes that tasks are ordered with decreasing priority.

Theorem 8. *A task set τ is schedulable with FP on a multiprocessor platform composed of m identical processors if for each task $\tau_k \in \tau$*

$$ \sum_{i < k} \min(\mathcal{W}_i(D_k), D_k - C_k + 1) < m(D_k - C_k + 1). \tag{10} $$

5 CONSIDERATIONS

The effectiveness of the schedulability conditions given by Theorems 6, 7, and 8 is magnified in the presence of heavy tasks. One of the main differences between our work and the results presented in [6] and [7] lies in the term $D_k - C_k + 1$ in the minimum. This term directly derives from term $D_k - C_k + 1$ in Theorem 5. 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 the real interference. Since we do not want to overestimate the total interference, we must consider only the fraction of the workload that can actually interfere with task τ_k. When no task misses its deadline, this fraction is bounded by $D_k - C_k$.

Example 1. Consider a task set τ composed of three tasks, each one with a deadline equal to the period, to be scheduled with EDF on a platform composed by $m = 2$ identical processors: $\tau = \{(20, 30, 30); (20, 30, 30); (5, 30, 30)\}$. It can be verified that both GFB and the test proposed in [7] fail. Instead, using Theorem 7, we have that the amount of interference we can consider on τ_1 (or τ_2) can be bounded by $D_1 - C_1 + 1 = 11$. The upper bound on the total interference is therefore given by $\min(20, 11) + \min(5, 11) = 16$, which is less than $m(D_1 - C_1 + 1) = 22$. Similarly, for task τ_3, the bound is $\min(20, 25) + \min(20, 25) = 40$, which is less than $m(D_3 - C_3 + 1) = 52$, and the test is passed.

Even if the derived algorithms contribute an increasing the number of schedulable task sets that can be detected, it is possible to show that the absolute performances of these tests and of any other schedulability test with comparable complexity previously proposed [6], [7], [8], [9], [10], [11], [12], [13], [14] are still far from being tight. In Section 7, we will show that these algorithms reject many schedulable task sets among a randomly generated distribution.

As for what concerns our tests, the problem is mainly due to the imprecise computation of the carry-in contribution to the total interference. Basically, in the proofs of our results, we assumed that the carried-in job (in the EDF case) or the first job (in the general and FP cases) of the interfering tasks is as close as possible to its deadline. This means assuming that every interfering task τ_i is as well interfered for $D_i - C_i$ time units, which is an overly pessimistic assumption, as the next example shows.

Example 2. Consider a task set τ composed by $\tau_1 = (1, 1, 1)$ and $\tau_2 = (1, 10, 10)$ to be scheduled with EDF on $m = 2$ processors. When applying Theorem 6 to check the schedulability of the task set, we find a negative result, due to the positive interference imposed by τ_2 on τ_1. However, it is easy to see that the task set is schedulable on two processors.

A less trivial example can be found by adding two more tasks τ_3 and τ_4 equal to τ_2. The test of Theorem 6 still fails because it assumes that the light tasks can receive enough interference to be pushed close to their deadline and interfere with τ_1. However, it is possible to show that the task set is schedulable: when the deadlines

of τ_1 and τ_2 coincide, a job of τ_2 would need to be pushed forward for $D_2 - C_2 = 9$ time-units to interfere with τ_1. But the maximum interference that τ_2 can receive is lower, as can be seen using the upper bound on the interferences given by (8) with $k = 2$:

$$I_2 \leq \frac{\sum_{i \neq 2} I_{i,2}}{m} \leq \frac{\sum_{i \neq 2} W_i\left(r_2^{j*}, d_2^{j*}\right)}{m} \leq \frac{10 + 1 + 1}{2} = 6.$$

Therefore, τ_2, as well as τ_3 and τ_4, will never be able to interfere with τ_1, and the task set is schedulable.

To improve the performances of our test, a tighter estimation of the interference imposed by a task τ_i on a task τ_k is needed. The following section will formally describe an iterative approach to overcome the drawbacks of the schedulability tests presented in Section 4.

6 ITERATIVE TEST

The technique used in Example 2 to prove that the proposed task set is schedulable suggests an iterative method to improve the estimation of the carry-in of an interfering task. By calculating the maximum interference a task τ_k can be subjected to, it is possible to know how close to its deadline a job J_k^j can be pushed. The lower the interference, the higher is the distance between the job finishing time f_k^j and its deadline d_k^j. We call this difference *slack* S_k^j of job J_k^j: $S_k^j = d_k^j - f_k^j$. The *slack* S_k of task τ_k is instead the minimum slack among all jobs of τ_k: $S_k = \min_j(d_k^j - f_k^j)$.

If the slack of a task τ_k is known, then it is possible to improve the estimation of the interference that τ_k can impose on other tasks. A positive slack will allow one to consider a less pessimistic situation than the one depicted in Figs. 2 and 3. When $S_k > 0$, the densest possible packing of jobs of τ_k will produce a lower workload in the considered interval. In this case, a tighter upper bound on the interference can be used, with beneficial effects on the schedulability analysis.

As before, we will first derive a general condition that is valid for any work-conserving scheduling algorithm, adapting it later to the EDF and FP cases.

6.1 Iterative Test for General Scheduling Algorithms

Before introducing slack values to tighten the schedulability conditions, we first show how to compute these terms for a given task set. However, computing the minimum slack time of a task in a multiprocessor system is not as easy as it is for classic uniprocessor systems. The next theorem shows a relation between the slack of a task τ_k and the interferences imposed by other tasks on it.

Theorem 9. *The slack of a task τ_k scheduled on a multiprocessor platform composed by m identical processors is given by*

$$S_k = D_k - C_k - \left\lfloor \frac{\sum_{i \neq k} \min(\overline{I}_{i,k}, D_k - C_k + 1)}{m} \right\rfloor \quad (11)$$
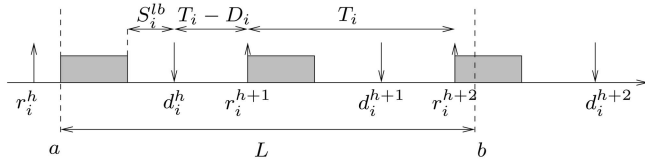
when (11) is positive.

Fig. 4. Densest possible packing of jobs of τ_i, when S_i^{lb} is a safe lower bound on the slack of τ_i.

Proof. When the right-hand term of (11) is positive, $\left\lfloor \frac{\sum_{i \neq k} \min(\overline{I}_{i,k}, D_k - C_k + 1)}{m} \right\rfloor \leq D_k - C_k$. Since $x < \lfloor x \rfloor + 1$: $\sum_{i \neq k} \min(\overline{I}_{i,k}, D_k - C_k + 1) < m(D_k - C_k + 1)$. Applying Lemma 4, we have $\overline{I}_k < (D_k - C_k + 1)$. Lemma 1 then gives $\overline{I}_{i,k} \leq \overline{I}_k < (D_k - C_k + 1)$. Therefore,

$$\min(\overline{I}_{i,k}, D_k - C_k + 1) = \overline{I}_{i,k}. \qquad (12)$$

Now, remember that J_k^{j*} is the job that suffers the maximum interference among all jobs of τ_k. From the definition of slack, it follows that $S_k = \min_j(d_k^j - f_k^j) = d_k^{j*} - f_k^{j*} = (r_k^{j*} + D_k) - (r_k^{j*} + C_k + \overline{I}_k)) = D_k - C_k - \overline{I}_k$. From the integer-time assumption, $\overline{I}_k = \lfloor \overline{I}_k \rfloor$. Using Lemma 3 and (12), $S_k = D_k - C_k - \lfloor \overline{I}_k \rfloor = D_k - C_k - \left\lfloor \frac{\sum_{i \neq k} \overline{I}_{i,k}}{m} \right\rfloor = D_k - C_k - \left\lfloor \frac{\sum_{i \neq k} \min(\overline{I}_{i,k}, D_k - C_k + 1)}{m} \right\rfloor$, proving the theorem.

To make use of the above result, we need to compute each interference term $\overline{I}_{i,k}$. Since we are not able to perform this computation in a reasonable amount of time, we will instead use an upper bound on $\overline{I}_{i,k}$ by exploiting the bounds we derived in Section 4. For task systems scheduled with a work-conserving algorithm, we have $\mathcal{W}_i(D_k) \geq \overline{I}_{i,k}$. A lower bound S_k^{lb} on the slack S_k of a task τ_k is then given by

$$S_k^{lb} = D_k - C_k - \left\lfloor \frac{\sum_{i \neq k} \min(\mathcal{W}_i(D_k), D_k - C_k + 1)}{m} \right\rfloor, \quad (13)$$

where $\mathcal{W}_i(D_k)$ is given by (6).

When a lower bound on the slack of a task τ_i is available, it is possible to give an even tighter upper bound on the interference τ_i can cause and use this information either when checking the schedulability of other tasks or when computing their slack parameters. If the value S_i^{lb} is positive, every job of τ_i will complete at least S_i^{lb} time units before its deadline. An upper bound on the maximum possible workload of τ_i in an interval of length L can then be derived by analyzing the situation in Fig. 4, which represents a less pessimistic situation than the one in Fig. 2. When a lower bound on the slack value of τ_i is known, we override the expression of $\mathcal{W}_i(L)$ as follows:

$$\mathcal{W}_i(L, S_i^{lb}) = N_i(L, S_i^{lb})C_i + \min\left(C_i, L + D_i - C_i - S_i^{lb} \right. \\ \left. - N_i(L, S_i^{lb})T_i\right), \qquad (14)$$

with

$$N_i(L, S_i^{lb}) = \left\lfloor \frac{L + D_i - C_i - S_i^{lb}}{T_i} \right\rfloor. \qquad (15)$$

Note that when a lower bound on S_i is not known, we can simply use $S_i^{lb} = 0$. In this case, (14) and (15) reduce to the original expressions given by (6) and (5). With these conventions, the next theorem follows from Theorem 9.

Theorem 10. *A lower bound on the slack of a task τ_k scheduled on a multiprocessor platform composed by m identical processors is given by*

$$S_k^{lb} = D_k - C_k - \left\lfloor \frac{\sum_{i \neq k} \min(\mathcal{W}_i(D_k, S_i^{lb}), D_k - C_k + 1)}{m} \right\rfloor \qquad (16)$$

when this term is positive.

Theorem 10 allows deriving an iterative method to check the schedulability of a task set scheduled with a work-conserving global scheduling algorithm on a multiprocessor platform:

- For every task in the task set, a lower bound value on the slack of the task is created and initially set to zero.
- Equation (16) is then used to compute a new value of the slack lower bound of the first task, with $\mathcal{W}_i(D_1, S_i^{lb})$ and $N_i(D_1, S_i^{lb})$ given by (14) and (15). If the computed value is positive, the upper bound is accordingly updated. If it is negative, the value is left to zero, and the task is marked as "temporarily not schedulable."
- The previous step is repeated for every task in the task set.
- If no task has been marked as temporarily not schedulable, the task set is declared *schedulable.* Otherwise, another round of slack updates is performed using the slack lower bounds derived at the previous cycle. If during a complete round no slack is updated, the iteration stops, and the task set is declared *not schedulable.*

Basically, if it is not possible to derive a positive lower bound on the slack for a task τ_k using (16), this task will be temporarily set aside, waiting for a slack update (i.e., increase) of potentially interfering tasks; if no update takes place during a complete iteration for all tasks in the system, then there is no possibility for further improvements, and the test fails. Otherwise, a higher slack value of a task τ_i could result in a sufficiently tighter upper bound on the interference on τ_k so that the schedulability of τ_k could now be positively verified. Since $\mathcal{W}_i(L, S_i^{lb})$ is a nonincreasing function of S_i^{lb}, the convergence of the algorithm is guaranteed.

A more formal version of the schedulability algorithm is given by procedure SCHEDULABILITYCHECK in Fig. 5. For now, suppose `Nround_limit` $= \infty$. When neither EDF nor FP scheduling algorithms are used, procedure SLACKCOMPUTE(τ_k) should select the slack lower bound value given by (16). The iteration continues updating the lower bounds on the slack values until either no more

$\text{SCHEDULABILITYCHECK}(\tau)$

  ▷ Check the schedulability of task set τ.
  $S_k^{lb} \leftarrow 0 \ (\forall k)$, Updated $\leftarrow$ true, Nround $\leftarrow 0$.
1 **while** (Updated && Nround < Nround_limit)
2 **do**
3   Feasible $\leftarrow$ true
4   Updated $\leftarrow$ false
5   **for** $k \leftarrow 1$ **to** n
    ▷ Try to update S_k^{lb}.
6   NewBound $\leftarrow \text{SLACKCOMPUTE}(\tau_k)$
7   **if** (NewBound < 0) Feasible $\leftarrow$ false
8   **else if** (NewBound > S_k^{lb})
9   { $S_k^{lb} \leftarrow$ NewBound
10   Updated $\leftarrow$ true }
   end for
11 Nround $++$
   ▷ When no task is infeasible, τ is schedulable.
12 **if** (Feasible) **return true**
  done
  ▷ Stop when no slack can be updated.
13 **return false**

Fig. 5. Iterative schedulability test for work-conserving scheduling algorithms.

update is possible or every task has been verified to be schedulable.

6.2 Iterative Test for EDF

When tasks are scheduled with EDF, it is possible to use a tighter bound on the interference $\overline{I}_{i,k}$. As in Section 4.3, we will consider the worst case workload produced by an interfering task τ_i when it has an absolute deadline coincident to a deadline of τ_k, and every other instance of τ_i is executed at the latest possible instant.

When a lower bound on the slack of τ_i is known, the upper bound on $\overline{I}_{i,k}$ given by (8) can be tightened. Consider the situation in Fig. 7. We express the workload of τ_i in $[r_k^{j*}, d_k^{j*})$ separating the contributions of the first job of τ_i a having deadline inside the considered interval from the contributions of later jobs of τ_i. There are $\left\lfloor \frac{D_k}{T_i} \right\rfloor$ later jobs, each one contributing for an entire worst-case computation time. Instead, the first job contributes for $\max\left(0, D_k - S_k^{lb} - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i\right)$, when this term is lower than C_i. We therefore obtain the following expression:

$$\overline{I}_{i,k} \le \left\lfloor \frac{D_k}{T_i} \right\rfloor C_i + \min\left(C_i, \left(D_k - S_i^{lb} - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i\right)_0\right) \quad (17)$$
$$\doteq \mathcal{I}_{i,k}\left(S_i^{lb}\right).$$

Again, when a lower bound on S_i is not known, we can simply use $S_i^{lb} = 0$. In this case, (17) reduces to (8). The next theorem then follows from Theorem 9.

Theorem 11. *A lower bound on the slack of a task τ_k scheduled with EDF on a multiprocessor platform composed by m identical processors is given by*

$\text{SLACKCOMPUTE}(\tau_k)$

  ▷ Computes a lower bound on the slack of τ_k,
  depending on the scheduling algorithm in use
1 **case** (EDF) : Bound $\leftarrow$ Equation (18)
2 **case** (FP) : Bound $\leftarrow$ Equation (19)
3 **else :** Bound $\leftarrow$ Equation (16)
4 **return** (Bound)

Fig. 6. Function computing the proper slack lower bound for EDF, FP, and general work-conserving schedulers.

$$S_k^{lb} = D_k - C_k - \left\lfloor \frac{1}{m} \sum_{i \ne k} \min\left(\mathcal{I}_{i,k}\left(S_i^{lb}\right), D_k - C_k + 1\right) \right\rfloor \quad (18)$$

when this term is positive.

For EDF-scheduled tasks, (18) allows deriving a slack lower bound tighter than the one given by (16).

The iterative method we described for general work-conserving algorithms applies as well to the EDF case. The only difference lies at line 1 of procedure $\text{SLACKCOMPUTE}(\tau_k)$ in Fig. 6, where the bound given by the left-hand side of (18) is selected for EDF-scheduled systems.

6.3 Iterative Test for FP

Since when using FP scheduling, the interference caused by lower priority tasks is always null, Theorem 10 can be modified by limiting the sum of the interfering terms to the higher priority tasks. Assuming tasks are ordered with decreasing priority, the next theorem follows:

Theorem 12. *A lower bound on the slack of a task τ_k scheduled with FP on a multiprocessor platform composed by m identical processors is given by*

$$S_k^{lb} = D_k - C_k - \left\lfloor \frac{\sum_{i<k} \min\left(\mathcal{W}_i\left(D_k, S_i^{lb}\right), D_k - C_k + 1\right)}{m} \right\rfloor \quad (19)$$

when this term is positive.

To apply the previously described iterative method to systems scheduled with FP, we can still use procedure $\text{SCHEDULABILITYCHECK}(\tau)$. In this case, the function $\text{SLACKCOMPUTE}(\tau_k)$ will select the bound given at line 2. However, there is an important difference from the EDF and the general case: for FP systems, when a task is found to be temporarily not schedulable during the first iteration, the test can immediately stop and return a false value. In fact, there is no hope that this result could be improved with successive tighter estimations of the interferences produced by lower priority tasks. In other
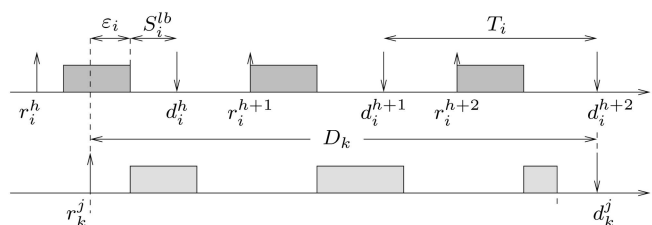

Fig. 7. Scenario with the maximum possible interference of τ_i on a job of τ_k with EDF, when S_i^{lb} is a safe lower bound on the slack of τ_i.

words, suppose that procedure SLACKCOMPUTE(τ_k) returns a negative slack lower bound for a task τ_k. The considered contribution to the total interference given by tasks with priority lower than τ_k's is null. Since the slack values are updated in order of task priority starting from the highest priority task, we know that later slack updates cannot further reduce the interference on τ_k or on higher priority tasks. Therefore, later calls to function SLACKCOMPUTE(τ_k) will always return the same negative value, eventually failing the test.

This observation allows limiting the number of slack updates to one for each task. This can be done by choosing Nround_limit = 1 in procedure SCHEDULABILITYCHECK(τ) when the scheduler is FP. The result will be a significant reduction in the complexity of the schedulability test.

7   EXPERIMENTAL RESULTS

In this section, we compare the tests derived in this paper with the best existing tests for the schedulability analysis of global scheduling algorithms for identical multiprocessor platforms.

For the EDF case, we will consider the following schedulability algorithms:

- the linear complexity test in [6] in the modified version for constrained deadline systems given by Theorem 3 (GFB),
- the $O(n^3)$ test described in [13] (BAK),
- our first EDF schedulability test in Theorem 7 (BCL EDF), and
- the iterative test given by procedure SCHEDULABILITYCHECK(τ) in Fig. 5 when EDF is used (I-BCL EDF).

Among schedulability tests for FP, we will instead compare the following algorithms, assuming that the DM priority assignment is used:

- the linear complexity test (derived in [11]) given by the density bound of Theorem 4 (DB),
- the $O(n^3)$ test described in [14] (BC),
- our first FP test in Theorem 8 (BCL FP),
- the iterative test given by procedure SCHEDULABILITYCHECK(τ) in Fig. 5 for FP systems (I-BCL FP).

For general work-conserving schedulers, we are not aware of any previously proposed schedulability test. We will therefore show only the behavior of our tests given by Theorem 6 (BCL) and by procedure SCHEDULABILITYCHECK(τ) when no useful information on the scheduler is available (I-BCL).

As a last term of comparison, we decided to compute as well the number of task sets that pass the load-based sufficient feasibility test in [23] (FB). Task systems passing this test are feasible on a given multiprocessor platform, meaning that there exist *at least one* scheduling algorithm that is able to meet every deadline. However, no information is given on which algorithm can be used to successfully schedule the task set, limiting the practical application of such a test. Remember instead that BCL detects task sets

that are schedulable with *any* (work-conserving) scheduling algorithm, which is a stronger claim.

We applied all the above tests to a randomly generated distribution of task sets. The simulations have been performed varying the number of processors, the number of tasks, and the total system utilization.

Every task has been generated in the following way: Utilization was extracted according to an exponential distribution with mean $\sigma_u = 0.25$, reextracting tasks with utilization $U_i > 1$; the period (and, implicitly, execution time) was from a uniform distribution in [0, 2,000]; and the deadline was from a uniform distribution between C_i and P_i.

For each experiment, we generated 1,000,000 task sets according to the following procedure:

1. Initially, we extract a set of $m + 1$ tasks.
2. We then verify if the generated task set passes a necessary condition for feasibility proposed in [24].
3. If the answer is positive, we test all the above-mentioned schedulability algorithms for the generated set. Then, we create a new set by adding a new task to the old set and return to the previous step.
4. When the necessary condition for feasibility in [24] is not passed, it means that no scheduling algorithm could possibly generate a valid schedule. In this case, the task set is discarded, returning to the first step.

This method allows generating task sets with a progressively higher number of elements, until the necessary condition for feasibility is violated.

The results are shown in the following histograms. Each line represents the number of task sets proved schedulable by one specific test. The curves are drawn connecting a series of points, each one representing the collection of task sets that have a total utilization in a range of 4 percent around the point. To give an *upper bound* on the number of feasible task sets, we included a continuous curve labeled with TOT, representing the distribution of valid task sets extracted, i.e., the number of generated task sets that meets the necessary condition for multiprocessor feasibility in [24]. To help in understanding the relative performances of the various algorithms, *keys are always ordered according to the total number of task sets detected by the corresponding test*: tests with a lower key position detect a lower number of task sets.

In Fig. 8, we show the case with $m = 2$ processors. In Fig. 8a, we plotted the number of task sets detected by all EDF-based schedulability tests (GFB, BAK, BCL EDF, and I-BCL EDF), while Fig. 8b represents all tests for FP (DB, BC, BCL FP, and I-BCL FP). In both histograms, we included the curves of the two tests that are applicable to any work-conserving scheduler (BCL and I-BCL), as well as the curves corresponding to the feasibility tests: the sufficient one (FB) and the necessary one (TOT).

The test that gives the best performances among EDF-based schedulability tests is I-BCL EDF: It significantly outperforms every existing schedulability test for EDF. For utilizations higher than 0.5, I-BCL EDF detects more than twice the task sets detected by GFB, which is in this case the best test among the existing ones. BAK and BCL EDF have much lower performances, comparable to the performances of the general tests valid for any work-conserving scheduler (BCL and I-BCL). Less than 1 percent of the generated task
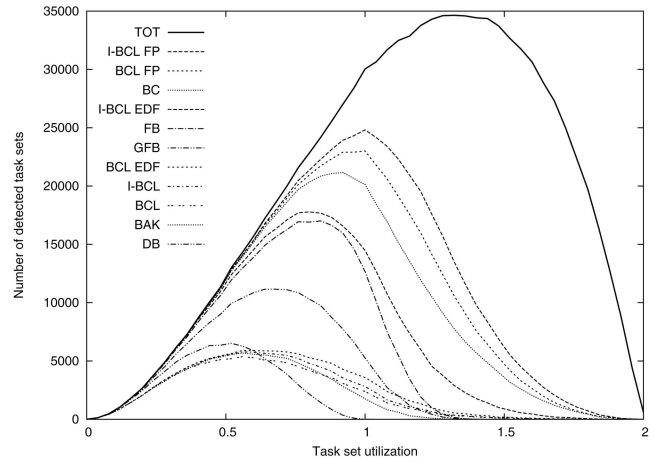
(a)
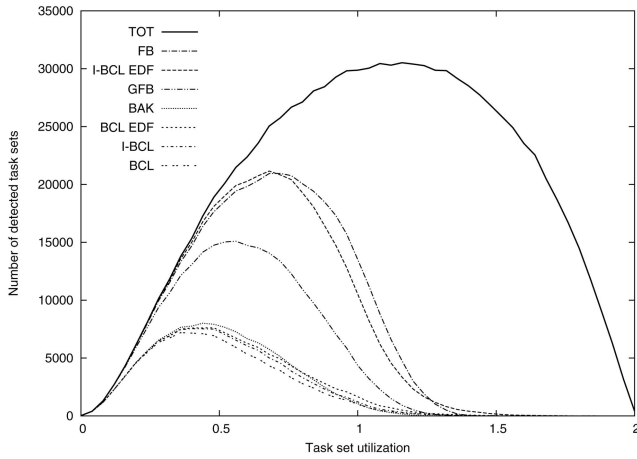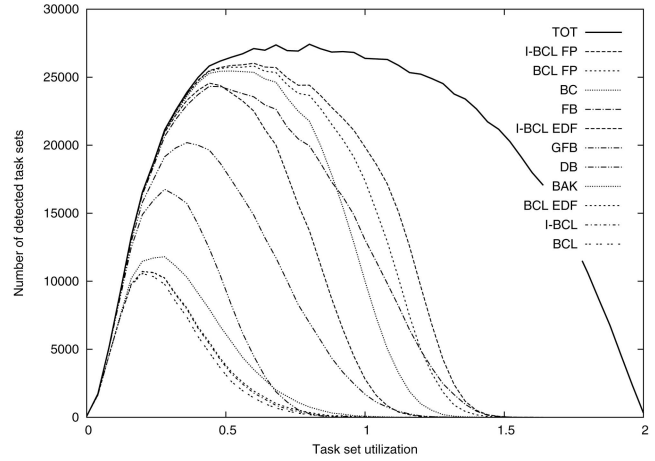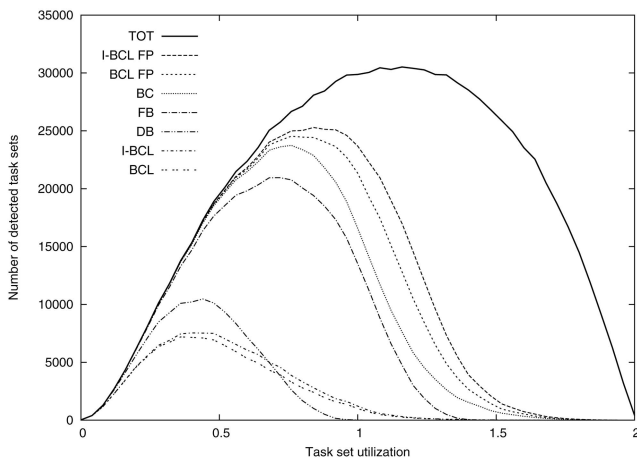
(b)

Fig. 8. Experiment with two processors and $\sigma_u = 0.25$ for (a) EDF and (b) FP.

(a)

(b)

Fig. 9. Experiments with two processors for (a) $\sigma_u = 0.10$ and (b) $\sigma_u = 0.50$.

sets is found to be schedulable by an existing algorithm for EDF but not by I-BCL EDF. The huge gap between I-BCL EDF and BCL EDF shows the power of the iterative approach that at each round refines the estimation of the slack values. Note that as anticipated in Section 3, BAK does not dominate GFB when deadlines are different than periods.

It is worth noting that I-BCL EDF almost detects as many task sets as FB. Remember that FB gives just a sufficient feasibility condition, without giving any information on which scheduler could effectively produce a valid schedule for the given task set. I-BCL EDF can instead detect a comparable number of schedulable task sets, knowing that EDF can be used to schedule them.

Looking at Fig. 8b, it is possible to see that with FP, the results are even better. This time I-BCL FP outperforms even FB. Considering the lower complexity of the FP version of our iterative test, this is a very interesting result. The next best test is BCL FP, which is much closer to the iterative version of the test than in the EDF case. This is due to the limitation `Nround_limit = 1` when a FP scheduler is used, as we explained in Section 6.3. Regarding other tests, less than 0.5 percent of the generated task sets is found to be schedulable by an existing algorithm for FP but not by I-BCL FP. BC has a fairly good behavior but has a higher complexity, as we will show later on.

Changing the mean utilization of the generated task sets, the results are similar to the above cases. In Fig. 9, we show the cases with $\sigma_u = 0,10$ and $\sigma_u = 0.50$, for all general, EDF, and FP schedulability tests. Even if the shape of the curves slightly changes, the relative ordering of the tests in terms of schedulability performances remains more or less the same.

Fig. 10a presents the case with four processors. The situation is more or less the same as before. The higher distance from the TOT curve is motivated by the worse performances of EDF and DM when the number of processor increases and does not seem to be a weak point of our analysis. The algorithms that suffer the most significant losses are GFB, FB, BAK, and DB. Further increasing the number of processors to $m = 8$ and $m = 16$, the above results are magnified, and our iterative algorithms have always the best performance, as shown in Figs. 10b and 10c. Note that to reduce the complexity of the simulations for the cases with 8 and 16 processors, we did not consider the most time-consuming or least performing tests (FB, BAK, and DB). For the same reason, we replaced the necessary condition for feasibility used in the task generation phase (the pseudopolynomial test in [24]) with a simpler condition: $U_{tot} \leq m$. This weaker condition causes a change in the shape of the TOT curve, accepting a larger number of unschedulable task sets.
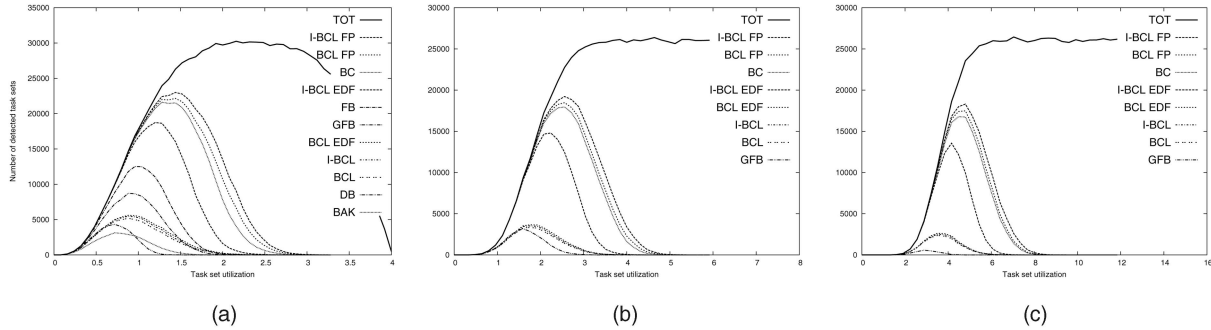
Fig. 10. Experiments with $\sigma_u = 0.25$ for (a) 4, (b) 8, and (c) 16 processors.

8 COMPUTATIONAL COMPLEXITY

The schedulability tests of Theorems 6, 7, and 8 are composed by n inequalities, each one requiring a sum of n terms. The overall complexity is therefore $O(n^2)$.

Instead, the complexity of the iterative tests introduced in Section 6 depends on the number of times the lower bound on the slack of a task can be updated. Consider procedure SCHEDULABILITYCHECK in Fig. 5. For now, suppose Nround_limit $= \infty$. A single invocation of function SLACKCOMPUTE takes $O(n)$ steps. Since the *for* cycle at line 5 calls this function once for each task, the complexity of a single iteration of slack updates is $O(n^2)$. Now, the outer cycle is iterated as long as there is a change in one of the slack values. Since for the integer-time assumption, the slack lower bound of a task τ_k can be updated at most $D_k - C_k$ times, a rough upper bound on the total number of iterations of the *while* cycle at line 1 is $\sum_k (D_k - C_k) = O(nD_{max})$. Therefore, the overall complexity of the algorithm is $O(n^3 D_{max})$. Anyway, the complexity can be significantly reduced if the test is stopped after a finite number Nround_limit of iterations. If this is the case, the total number of steps taken by the schedulability algorithm is $O(n^2$Nround_limit$)$.

For the FP case, we know from Section 6.3 that setting Nround_limit $= 1$ does not degrade the schedulability performances of the test. For EDF and for the general case, instead, limiting the number of cycles to a small value could reduce the number of admitted task sets, rejecting some schedulable task set that could otherwise be admitted after a few more steps. However, the schedulability loss is negligible even with very low Nround_limit's. We performed experiments for different values of Nround_limit $: 1, 2, 3, \infty$. When the slack upper bound is updated at most once for each task, the behavior of procedure SCHEDULABILITYCHECK in the EDF case is almost identical to the test given by Theorem 7 (BCL EDF). When two updates for each task are allowed, the number of schedulable task sets found by the iterative algorithm increases dramatically. For Nround_limit $= 3$, the test detects almost every task set that can be detected using an unbounded Nround_limit. This means that using procedure SCHEDULABILITYCHECK with Nround_limit $= 3$ or slightly higher values, we obtain an efficient solution to detect a high number of schedulable task sets at low computational effort. The low complexity $(O(n^2))$ of the test suggests the application of this algorithm to systems with very strict timely requirements and for runtime admission control.

9 CONCLUSIONS

We developed a new schedulability analysis of real-time systems globally scheduled on a platform composed by identical processors. We presented sufficient schedulability algorithms that are able to check in polynomial and pseudopolynomial time whether a periodic or sporadic task set can be scheduled on a multiprocessor platform. The tests we proposed vary in terms of computational complexity and number of schedulable task sets detected. Our experiments show that the iterative algorithm given by procedure SCHEDULABILITYCHECK detects the highest number of schedulable task sets among all existing tests. Only a negligible percentage of task sets is detected by some algorithm in the literature but not by our iterative test. This improvement in terms of schedulable task sets detected is given at a low computational cost. This consideration suggests the use of our iterative test also for runtime admission control.

Even if we contributed to significantly increase the number of schedulable task sets that can be detected at a reasonable computational effort, we still do not know how big is the gap from a hypothetical necessary and sufficient schedulability condition. Note that the TOT curve in our simulations *does not* represent the number of task set schedulable with EDF or FP, nor does it indicate how many task sets are feasible. If an exact feasibility test existed, its curve would be below the TOT curve. A necessary and sufficient schedulability test for EDF or for FP would have an even lower curve. While an exponential-time exact schedulability test for strictly periodic FP systems has been proposed in [25], we are not aware of any such test for sporadic task sets with deadlines different than periods.

Since the provable superiority of EDF in the uniprocessor case cannot be generalized to multiprocessor platforms, there is no known reason why EDF should be used for nonpartitioned approaches.[2] A simpler FP scheduler could have similar schedulability performances at a lower implementation cost. Moreover, it is widely known that Real-Time system developers are interested in high scheduling performances at least as much as they are interested in predicting if a deadline will be missed with such schedulers. Therefore, using an allegedly better scheduling algorithm that does not come with a good schedulability test is probably worse than relying on a simple FP scheduler

---

2. For partitioned systems, the optimality of EDF as a local scheduler continues to be valid.

that can take advantage of a good test. We showed that the schedulability test we proposed for FP systems (I-BCL FP) detects the highest number of schedulable task sets among the existing schedulability algorithms for globally scheduled multiprocessor systems. Using a simple FP scheduler in combination with our schedulability algorithm seems to be a good solution for guaranteeing the hard real-time constraints of a given application.

When a FP scheduler is used, an open question is which priority assignment allows scheduling the highest number of task sets. In our experiments, we used DM. An interesting task could be to explore which priority assignment could further magnify the performances of the I-BCL FP schedulability test. For example, an option could be to single out the heaviest tasks by assigning them higher priorities and scheduling the light tasks with DM. In this way, tasks having tighter timely requirements can execute at a privileged level, without being interfered by the other tasks. We intend to analyze this issue in future works, together with the analysis of more general scheduling algorithms, like hybrid or dynamic-job-priority schedulers, which are expected to have a lower gap from a necessary and sufficient feasibility condition.

Another factor that we intend to include into our analysis is the blocking time due to the exclusive access to shared resources. Thanks to the intuitive form of the slack-based test we presented, we believe that an extension of our tests to account as well for the blocking times can be easily derived for the most used shared resource protocols.

Further improvements on our schedulability analysis are possible. One of the potential drawbacks of the approach we followed consists of assuming all tasks being always maximally interfered. This is an overly pessimistic assumption that we introduced to simplify the final test. We have ideas on how to refine the estimation of the relative interferences among the various tasks, increasing the complexity of the schedulability tests. Anyway, we believe that the algorithms we proposed here are a good compromise between the number of schedulable task sets detected and the overall computational cost.

REFERENCES

[1] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.
[2] J.M. Calandrino, J.H. Anderson, and D.P. Baumberger, "A Hybrid Real-Time Scheduling Approach for Large-Scale Multicore Platforms," *Proc. 19th Euromicro Conf. Real-Time Systems (ECRTS),* 2007.
[3] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," *Algorithmica,* vol. 15, no. 6, pp. 600-625, June 1996.
[4] J. Anderson and A. Srinivasan, "Pfair Scheduling: Beyond Periodic Task Systems," *Proc. Seventh Int'l Conf. Real-Time Computing Systems and Applications (RTCSA '00),* Dec. 2000.
[5] P. Holman and J.H. Anderson, "Adapting Pfair Scheduling for Symmetric Multiprocessors," *J. Embedded Computing,* vol. 1, no. 4, pp. 543-564, 2005.
[6] J. Goossens, S. Funk, and S. Baruah, "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors," *Real Time Systems,* vol. 25, nos. 2-3, pp. 187-205, 2003.
[7] T. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis," *Proc. 24th IEEE Real-Time Systems Symp. (RTSS '03),* pp. 120-129, Dec. 2003.
[8] B. Andersson, S. Baruah, and J. Jonsson, "Static-Priority Scheduling on Multiprocessors," *Proc. 22nd IEEE Real-Time Systems Symp. (RTSS '01),* pp. 193-202, Dec. 2001.
[9] B. Andersson, "Static-Priority Scheduling on Multiprocessors," PhD dissertation, Dept. of Computer Eng., Chalmers Univ., 2003.
[10] M. Bertogna, M. Cirinei, and G. Lipari, "Improved Schedulability Analysis of EDF on Multiprocessor Platforms," *Proc. 17th Euromicro Conf. Real-Time Systems (ECRTS '05),* pp. 209-218, July 2005.
[11] M. Bertogna, M. Cirinei, and G. Lipari, "New Schedulability Tests for Real-Time Tasks Sets Scheduled by Deadline Monotonic on Multiprocessors," *Proc. Ninth Int'l Conf. Principles of Distributed Systems (OPODIS '05),* Dec. 2005.
[12] T.P. Baker, "An Analysis of Fixed-Priority Schedulability on a Multiprocessor," *Real-Time Systems: The Int'l J. Time-Critical Computing,* vol. 32, nos. 1-2, pp. 49-71, 2006.
[13] T.P. Baker, "An Analysis of EDF Schedulability on a Multiprocessor," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 8, pp. 760-768, Aug. 2005.
[14] T.P. Baker and M. Cirinei, "A Unified Analysis of Global EDF and Fixed-Task-Priority Schedulability of Sporadic Task Systems on Multiprocessors," *J. Embedded Computing,* http://www.cs.fsu.edu/research/reports/TR-060401.pdf, 2007.
[15] S.K. Dhall and C.L. Liu, "On a Real-Time Scheduling Problem," *Operations Research,* vol. 26, pp. 127-140, 1978.
[16] M. Bertogna, "Real-Time Scheduling Analysis for Multiprocessor Platforms," PhD dissertation, Scuola Superiore Sant'Anna, 2008.
[17] A. Srinivasan and S. Baruah, "Deadline-Based Scheduling of Periodic Task Systems on Multiprocessors," *Information Processing Letters,* vol. 84, no. 2, pp. 93-98, 2002.
[18] S. Baruah, "Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Task Systems on Identical Multiprocessors," *IEEE Trans. Computers,* vol. 53, no. 6, June 2004.
[19] J. Anderson and A. Srinivasan, "Mixed Pfair/ERfair Scheduling of Asynchronous Periodic Tasks," *Proc. 13th Euromicro Conf. Real-Time Systems (ECRTS '01),* June 2001.
[20] B. Andersson and E. Tovar, "Multiprocessor Scheduling with Few Preemptions," *Proc. 12th IEEE Int'l Conf. Real-Time Computing Systems and Applications (RTCSA '06),* pp. 322-334, 2006.
[21] S. Cho, S.K. Lee, and K.-J. Lin, "On-Line Algorithms for Real-Time Task Scheduling on Multiprocessor Systems," *Proc. Fifth IASTED Int'l Conf. Internet and Multimedia Systems and Applications (IMSA '01),* pp. 395-400, Aug. 2001.
[22] M. Cirinei and T.P. Baker, "Edzl Scheduling Analysis," *Proc. 19th Euromicro Conf. Real-Time Systems (ECRTS '07),* July 2007.
[23] N. Fisher and S. Baruah, "The Global Feasibility and Schedulability of General Task Models on Multiprocessor Platforms," *Proc. 19th Euromicro Conf. Real-Time Systems (ECRTS '07),* July 2007.
[24] T.P. Baker and M. Cirinei, "A Necessary and Sometimes Sufficient Condition for the Feasibility of Sets of Sporadic Hard-Deadline Tasks," *Proc. 27th IEEE Real-Time Systems Symp. (RTSS '06),* pp. 178-190, 2006.
[25] L. Cucu and J. Goossens, "Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors," *Proc. 11th IEEE Int'l Conf. Emerging Technologies and Factory Automation (ETFA '06),* Sept. 2006.

Marko Bertogna received the degree (summa cum laude) in telecommunications engineering from the University of Bologna, Italy, in 2002 and the PhD degree in computer science from the Scuola Superiore Sant'Anna, Pisa, Italy, in 2008. He currently has a postdoc position at the Scuola Superiore Sant'Anna. In 2002, he visited TU Delft, Netherlands, working on optical devices. In 2006, he visited the University of North Carolina, Chapel Hill, working with Professor Sanjoy Baruah on scheduling algorithms for single and multicore real-time systems. His research interests include scheduling and schedulability analysis of real-time multiprocessor systems, protocols for the exclusive access to shared resources, and reconfigurable devices. He received the 2005 IEEE/Euromicro Conference on Real-Time Systems Best Paper Award.

Michele Cirinei received the laurea degree (summa cum laude) in computer science engineering from the University of Pisa, Pisa, Italy, in 2004 and the PhD degree in computer science, for his research on the use of multiprocessor platforms for real-time systems, from the Scuola Superiore Sant'Anna, Pisa, in 2007. His research particularly focused on how to improve both the computational power and fault tolerance of systems with strict time constraints. Since July 2007, he has moved to Arezzo, where he is a software designer at SECO, a European designer, manufacturer, and marketer of highly integrated board computers and systems for embedded applications.
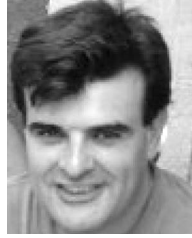
Giuseppe Lipari received the degree in computer engineering from the University of Pisa in 1996 and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna, Pisa, Italy, in 2000. Currently, he is an associate professor of operating systems with the Scuola Superiore Sant'Anna. His main research activities are in real-time scheduling theory and its application to real-time operating systems, soft real-time systems for multimedia applications, and component-based real-time systems. He has been a member of the program committees of many conferences in the field. He is currently an associate editor of the *IEEE Transactions on Computers*. He is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.