An Exact Schedulability Test for Global FP Using State Space Pruning

Artem Burmyakov*
Enrico Bini
Eduardo Tovar*

*CISTER Research Center

CISTER-TR-150503

1, Jan, 2015

Artem Burmyakov*, Enrico Bini, Eduardo Tovar*

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: armbv@isep.ipp.pt, emt@isep.ipp.pt

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

Abstract

We propose an exact schedulability test for sporadic real- time tasks with constrained deadlines, scheduled by Global Fixed Priority (GFP). Our test is faster and less mem- ory consuming than other state-of-the-art exact tests. We achieve such results by employing a set of techniques that cut down the state space of the analysis, which extend the prior work by Bonifaci and Marchetti-Spaccamela. Our test is implemented in C++ code, and it is publicly available.

An Exact Schedulability Test for Global FP Using State Space Pruning

Artem Burmyakov
CISTER/INESC-TEC, ISEP,
Polytechnic Institute of Porto,
Portugal

Enrico Bini
Scuola Superiore Sant'Anna,
Pisa, Italy

Eduardo Tovar
CISTER/INESC-TEC, ISEP,
Polytechnic Institute of Porto,
Portugal

ABSTRACT

We propose an exact schedulability test for sporadic real-time tasks with constrained deadlines, scheduled by Global Fixed Priority (GFP). Our test is faster and less memory consuming than other state-of-the-art exact tests. We achieve such results by employing a set of techniques that cut down the state space of the analysis, which extend the prior work by Bonifaci and Marchetti-Spaccamela. Our test is implemented in C++ code, and it is publicly available.

1. INTRODUCTION

The key property of a real-time system is time predictability: the operations in such a system must be performed within a strictly defined time. A scheduler is employed to allocate the available CPU time to pending jobs of real-time tasks, using certain priority rules.

At design time, a schedulability test is used to ensure that all system deadlines will be met at runtime. Although fast exact schedulability tests exist for a uniprocessor platform [1, 2, 3], existing exact tests for a multiprocessor platform are highly time- and memory-consuming. Hence, a considerable amount of research has focused on sufficient tests, which however significantly overestimate the system demand for processing capacities.

To understand the need for an exact test, we thoroughly evaluated the performance of the sufficient test of Guan *et al.* [4] against the exact test of Bonifaci and Marchetti-Spaccamela [5], and in certain cases, its pessimism exceeds 50%: more than half of those task sets, reported by Guan's test as unschedulable, are in fact schedulable. However, due to high computation time and memory consumption, Bonifaci's exact test becomes intractable even for small systems.

In this paper we derive a faster exact schedulability test for sporadic tasks with constrained deadlines.

1.1 Related works

The first exact schedulability test has been proposed by Baker and Cirinei [6], for several discrete-time schedulers. To check whether a given system is schedulable, the authors solve a reachability problem in a finite state transition graph: the algorithm traverses such a graph until it either finds a state with a violated deadline, or all feasible states are confirmed schedulable.

Bonifaci and Marchetti-Spaccamela [5] improved significantly Baker's test [6]. They have also refined the complexity bounds for the exact test, showing that it has polynomial space complexity, rather than exponential, as reported in [6]. An efficient C++ implementation for Bonifaci's test is publicly available[1]. As our work strongly relies on [5], Section 3 will provide more details on that work.

Another exact test was proposed by Geeraerts *et al.* [7], using formal verification methods. Both Bonifaci's test [5] and Geeraerts' test [7] apply to most of online discrete-time schedulers, such as GFP and GEDF, and allow tasks with arbitrary deadlines. Sun and Lipari [8] have instead derived a test specifically for GFP, by using a linear hybrid automaton.

Finally, Guan *et al.* [9] proposed a test for strictly periodic tasks, for fixed-priority scheduling, using model-checking. However, in multiprocessor scheduling, the scenario with periodic activations is not the worst-case for sporadic tasks, and an exact test for sporadic tasks must analyze a significantly larger number of legal release sequences.

Our evaluation has shown that Bonifaci's test [5] for GFP is faster, when compared to the exact tests using a timed automaton, that is by Geeraerts *et al.* [7], and Sun and Lipari [8]. Such a conclusion is based on comparing running times reported in [7], [8] against our evaluation reported in Section 5. The runtime gain of Bonifaci's test increases noticeably for task sets with a larger number of tasks, and a larger range of task periods. For example, while Geeraerts' test [7] is constrained to task periods not exceeding 6-8, Bonifaci's test can deal with larger task periods up to 40. We have also made some initial evaluation of constraint programming and global optimization methods (such an optimization problem can be formulated through the notation proposed in Section 2), but the resulted runtime was much longer than for Bonifaci's test [5]. For these reasons, we have chosen Bonifaci's test [5] as an initial ground to apply our improvements. Anyway, we remark that all runtime reduction techniques derived in this work can be applied to any other existing exact test for GFP.

1.2 Contributions of this paper

First, we estimate the pessimism of sufficient schedulability tests for GFP, and confirm the need for a better solution. Then, we derive an exact test, faster than Bonifaci's test [5], by exploiting (i) a constraint to maximize job interference, (ii) a sufficient schedulability constraint, (iii) a constraint for critical job release instants, and (iv) an optimized clock transition between checked states.

[1]`http://www.iasi.cnr.it/~vbonifaci/software.php`

2. DEFINITIONS

We model a real-time system by a set of sporadic tasks $\mathcal{T} = \{\tau_1, \dots, \tau_n\}$, wherein each task $\tau_i = (C_i, D_i, P_i)$ is characterized by an execution time C_i, a relative deadline D_i, and a minimum interarrival time P_i. We consider constrained deadlines $D_i \le P_i$. Each task $\tau_i \in \mathcal{T}$ generates a potentially infinite sequence of jobs, whose releases are separated by at least P_i. A job released by τ_i has an execution requirement of C_i time units, and a deadline at D_i time units after the job arrival time. All task parameters are assumed to be integers. Tasks are scheduled by Global Fixed Priorities (GFP) upon m identical processors, and are sorted by decreasing priorities. We consider that scheduling decisions are taken at discrete time instants $\mathbb{N} = \{0, 1, 2, \dots\}$.

To represent the possible scenarios of job releases, we define the *release sequence* R as a set of n functions

$$R = \{r_1(t), \dots, r_n(t) \mid t \in \mathbb{N}\},$$

wherein each function $r_i : \mathbb{N} \to \{0, 1\}$ is such that $r_i(t) = 1$ if τ_i releases a job at time t, and $r_i(t) = 0$ otherwise.

The release sequence R is said *legal* if the constraint on the job minimum interarrival times is met; that is:

$$\forall i = 1, \dots, n,\ \forall t_r, t'_r \in \mathbb{N},\ t_r < t'_r, \\ (r_i(t_r) = 1\ \wedge\ r_i(t'_r) = 1) \quad \Rightarrow \quad t'_r - t_r \ge P_i. \quad (1)$$

Also, we define the *finishing sequence* F as a set of n functions

$$F = \{f_1(t), \dots, f_n(t) \mid t \in \mathbb{N}\},$$

wherein each function $f_i : \mathbb{N} \to \{0, 1\}$ is such that $f_i(t) = 1$ if a job of τ_i is completed at time t, and $f_i(t) = 0$ otherwise.

Set Q is defined by

$$Q = \{q_1(t), \dots, q_n(t) \mid t \in \mathbb{N}\},$$

wherein each function[2] $q_i : \mathbb{N} \to \{0, 1\}$ indicates if τ_i has a pending job at time t (in the run queue), defined by:

$$q_i(t) = \sum_{t'=0}^{t} r_i(t') - \sum_{t'=0}^{t} f_i(t'). \quad (2)$$

A schedule is represented by a set S of n functions

$$S = \{s_1(t), \dots, s_n(t) \mid t \in \mathbb{N}\},$$

with $s_i(t) = 1$ if any processor among the m available ones is allocated to τ_i over time $[t, t+1)$, and $s_i(t) = 0$ otherwise. With these notations, GFP schedule S is formally defined by

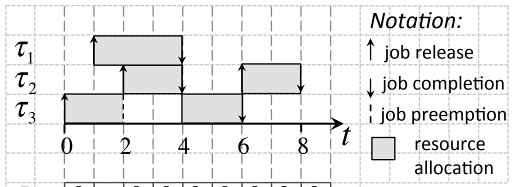
$$s_i(t) = 1 \qquad \Leftrightarrow \qquad q_i(t) > 0\ \wedge\ \sum_{\ell=1}^{i-1} q_\ell(t) < m, \quad (3)$$

and the indicator function $f_i(t)$ of the finishing time, as

$$f_i(t) = 1 \qquad \Leftrightarrow \\ \exists t_r < t :\ r_i(t_r) = 1 \wedge \sum_{t'=t_r}^{t-1} s_i(t') = C_i \wedge s_i(t-1) = 1. \quad (4)$$

Fig. 1 illustrates an example of a GFP schedule of $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$. Below the schedule, we list the respective values for R, F, Q, and S, where the i-th row corresponds to task τ_i, and the j-th column corresponds to time instant $t = j-1$.

[2]We assume that no deadline miss has occurred by time t, meaning that τ_i has at most one pending job, due to $D_i \le P_i$.



$$R: \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$F: \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

$$Q: \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array}$$

$$S: \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array}$$

Figure 1: GFP schedule

Time instants t_r, t_c are said to be the release and completion times of the same job of τ_i, if it holds:

$$r_i(t_r) = 1\ \wedge\ f_i(t_c) = 1\ \wedge\ \sum_{t=0}^{t_r} r_i(t) = \sum_{t=0}^{t_c} f_i(t), \quad (5)$$

meaning that i) some job of τ_i is released at time t_r, ii) some job of τ_i is completed at time t_c, and iii) the number of τ_i releases over time $[0, t_r]$ equals to the number of τ_i completions over time $[0, t_c]$.

We define schedulability of $\mathcal{T}$ as follows.

Definition 1 (Schedulability of $\mathcal{T}$). *Let $\mathbb{L}_R$ denote all legal release sequences of task set $\mathcal{T}$, satisfying (1). $\mathcal{T}$ is said schedulable upon m processors, if for any $R \in \mathbb{L}_R$, all jobs of task τ_i, $i = 1, \dots, n$, meet their deadlines:*

$$\forall R \in \mathbb{L}_R,\ \ \forall i \in \{1, \dots, n\},\ \ \forall (t_r,\ t_c), \qquad t_c - t_r \le D_i, \quad (6)$$

where t_r, t_c are the respective release and completion times for the same job of τ_i, defined by (5).

3. BACKGROUND ON EXACT SCHEDULABILITY TESTS

Bonifaci and Marchetti-Spaccamela [5] analyzed the schedulability of sporadic tasks by traversing a finite non-deterministic state transition graph, searching for a state with a violated deadline. As our work aims at improving their approach, next we revisit the main ideas behind Bonifaci's work [5].

At a given time t, the state of the set of tasks is modeled by

$$\big(c_i, d_i, p_i\big)_{i=1}^{n} \in \mathbb{N}^{3n}, \quad (7)$$

where $c_i \in \{0, \dots, C_i\}$ is the remaining execution time of τ_i pending job at t, if any; $d_i \in \{0, \dots, D_i\}$ is the remaining time until its deadline; and $p_i \in \{0, \dots, P_i\}$ is the remaining time until the earliest release of the next job of τ_i.

A state transition graph for $\mathcal{T}$ is constructed as follows (see also Fig. 2). Each state in the graph represents a system

Table 1: Bonifaci's test: Task set example

i	C_i	P_i	D_i
1	2	3	3
2	1	4	4
3	3	5	5

Algorithm 1 Exact schedulability test

```
 1: procedure EXACTSCHEDULABILITYTEST
 2:     $V \leftarrow \emptyset$                          ▷ initialize $V$
 3:     $G \leftarrow (0,0,0)_{i=1}^{n}$                  ▷ initialize $G$
 4:     while $G \neq \emptyset$ do
 5:         $g \leftarrow \mathsf{Dequeue}(G)$
 6:         compute $G'$ for $g$                     ▷ from Eq. (8)
 7:         for each $g' \in G'$ do
 8:             if $g' \notin V$ then
 9:                 if $\exists i$, (10) holds then    ▷ deadline miss
10:                     return Unschedulable
11:                 end if
12:                 $V = V \cup \{g'\}$
13:                 $G = \mathsf{Enqueue}(G, g')$
14:             end if
15:         end for
16:     end while
17:     return Schedulable
18: end procedure
```

state $(c_i, d_i, p_i)_{i=1}^{n}$ at a given time t. The initial state is $(0,0,0)_{i=1}^{n}$, meaning that no job has been yet released.

The state transition law, which governs the transition from state $g = (c_i, d_i, p_i)$ at time t to the next state $g' = (c'_i, d'_i, p'_i)$ at time $t+1$, is the following:

$$g' \in G' \iff \begin{cases} c'_i = c_i - s_i(t) + r_i(t+1)\, C_i \\ d'_i = \max(d_i - 1, 0) + r_i(t+1)\, D_i \\ p'_i = \max(p_i - 1, 0) + r_i(t+1)\, P_i, \end{cases} \tag{8}$$

where G' is a set of all successors for g at time $t+1$. In the equation above, $s_i(t)$ is the schedule function of τ_i uniquely determined by the system state g through (3), and $r_i(t+1)$ represents the release function (the "input" to the system) satisfying (1); that is

$$\begin{aligned} p_i - 1 > 0 \quad &\Rightarrow \quad r_i(t+1) = 0 \\ p_i - 1 = 0 \quad &\Rightarrow \quad r_i(t+1) \in \{0, 1\}. \end{aligned} \tag{9}$$

State $(c'_i, d'_i, p'_i)_{i=1}^{n}$, at time t, is a scheduling failure state if some job misses its deadline:

$$c'_i - r_i(t)\, C_i > d'_i - r_i(t)\, D_i, \tag{10}$$

with $r_i(t)$ defined by (9). This condition is true when the remaining execution time for a job (LHS of (10)) exceeds the remaining time until its deadline (RHS of (10)). $r_i(t)C_i$, $r_i(t)D_i$ are subtracted to correctly consider the case when the job deadline of τ_i coincides to the next release of τ_i.

Once the scheduling failure state is encountered, the algorithm reports unschedulability of $\mathcal{T}$, and terminates. If instead all feasible states have been checked, and no failure state has been detected, then $\mathcal{T}$ is reported schedulable.

Algorithm 1 implements Bonifaci's test using breadth-first search. It maintains two additional data structures: V is a set of checked states at previous iterations, and G is a FIFO queue, containing states for further examination. The set

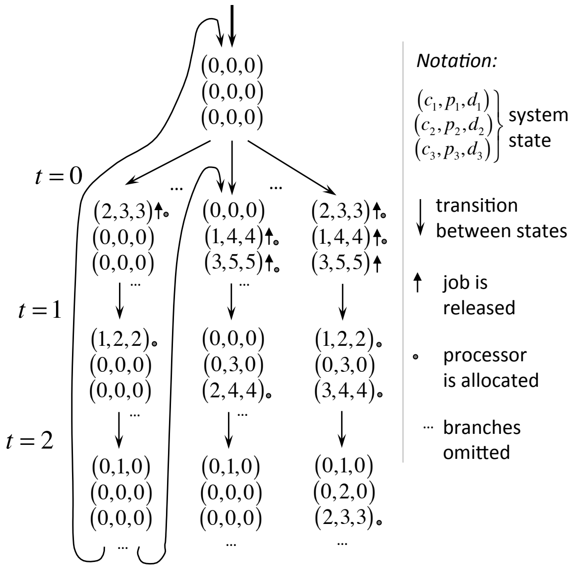


Figure 2: State transition graph

V is initially empty, and queue G contains only the initial state $g_0 = (0,0,0)_{i=1}^{n}$. At the first iteration of the while loop, the algorithm removes g_0 from G, and computes set G' of successors for g_0 using (8). Then, each state $g' \in G'$ is checked for a deadline miss (line 9), added to a list of checked states V (line 12), and added to queue G, to examine the g' successors at further iterations. At each iteration of the while loop, the algorithm removes the first state g from queue G (line 5), and computes the set G' of successors for g (line 6). Each state $g' \in G'$ that has not been checked yet (that is $g' \notin V$), is checked for a deadline miss, and added to V and G. The algorithm terminates when G becomes empty, meaning that all feasible system states were examined.

A higher runtime efficiency of Bonifaci's Algorithm 1 compared to other exact tests is mainly thanks to condition in line 8: each system state in a graph is checked only once. However, the same approach cannot be applied directly to tests using a timed automaton, due to specifics of a timed automaton. One solution has been proposed by Geeraerts et al. [7], who derived a so called simulation relation technique for a timed automaton, but it does not seem to be more efficient than Bonifaci's approach.

Anyway, Algorithm 1 has exponential time and polynomial space complexity, and it might not terminate in a reasonable time even for a small $\mathcal{T}$. According to our evaluation, Bonifaci's test is capable to deal with up to 5–6 tasks scheduled upon 2 processors, considering very small range of task periods, not exceeding 40.

Consider $\mathcal{T}$ with the parameters reported in Table 1, to be scheduled by GFP upon $m = 2$ processors. In Fig. 2 we report fragments of the state transition graph for such $\mathcal{T}$. The total number of states in the full graph is 191.

For the same $\mathcal{T}$, our test checks 12 states only, instead of 191, in one sixth of the running time relative to [5], and the efficiency of our test increases for larger task sets.

4. A FASTER SCHEDULABILITY TEST

In this section we derive a faster exact schedulability test for $\mathcal{T}$. Below we test schedulability of τ_k, assuming that $\tau_1, \ldots, \tau_{k-1}$ are schedulable.

4.1 Job interference

We first show that, when analyzing the schedulability of τ_k, we can safely ignore any job of a higher-priority task τ_i, $i < k$, that causes no interference to any lower-priority tasks $\tau_{i+1}, \ldots, \tau_k$.

Let us define job interference as follows.

DEFINITION 2 (JOB INTERFERENCE). *Let J_i denote an arbitrary job of τ_i, with release and completion times denoted by t_r and t_c respectively. Job J_i is said to interfere with a lower priority job J_ℓ of τ_ℓ, $\ell > i$, if at some time $t \in [t_r, t_c)$ a processor is allocated to J_i, but not to J_ℓ:*

$$\begin{aligned}&\exists \ell > i, \quad \exists t \in [t_r, t_c) : \\ &\qquad s_i(t) = 1 \ \wedge\ q_\ell(t) = 1 \ \wedge\ s_\ell(t) = 0,\end{aligned} \tag{11}$$

with $s_i(t)$, $q_\ell(t)$ defined by (3) and (2), respectively.

We clarify this definition on an example. Let $\mathcal{T} = \{\tau_1, \ldots, \tau_4\}$ be scheduled upon $m = 2$ processors. Consider the release sequence R for $\mathcal{T}$ as depicted in Fig. 3a, and suppose that we analyze schedulability of task τ_4. Job $J_{1,1}$ of τ_1 interferes with job $J_{3,1}$ of τ_3 at time $t = 2$, as (11) holds:

$$s_1(2) = 1 \ \wedge\ q_3(2) = 1 \ \wedge\ s_3(2) = 0.$$

Instead, job $J_{3,1}$ does not interfere with $J_{4,1}$, as (11) is violated. By removing such non-interfering jobs, we can produce a different arrival sequence that does not affect the schedule of task τ_4. Let us transform R, depicted in Fig. 3a, into R', by erasing all jobs of task τ_i, $i < 4$, which do not interfere with lower priority jobs. These jobs are $J_{2,2}$, $J_{3,1}$, and $J_{3,2}$. Observe that the amount of resource available for τ_4 in R' is the same as in R.

THEOREM 1. *Assume that $\tau_1, \ldots, \tau_{k-1}$ are schedulable. Let $R = \{r_1(t), \ldots, r_k(t)\}$ be any legal release sequence for $\mathcal{T}$, and let $J_{i,t}$ denote the τ_i job, released at time t. Let R' be a new release sequence that excludes all jobs of task τ_i from R, $i < k$, which violate the interference condition (11):*

$$\begin{aligned} R' =& \{r'_1(t), \ldots, r'_k(t)\} : \\ & r'_i(t) = \begin{cases} 1, & \text{if } r_i(t) = 1 \text{ and (11) holds for } J_{i,t} \\ 0, & \text{otherwise} \end{cases}, \\ & \qquad i = 1, \ldots, k-1, \\ & r'_k(t) = r_k(t). \end{aligned} \tag{12}$$

Then, the job of task τ_k pending at time t (if exists) misses its deadline in R iff it misses its deadline in R'.

The proof of Theorem 1 is provided in the Appendix.

According to Theorem 1, the worst-case release sequence for τ_k is among those satisfying the following condition.

COROLLARY 1. *Let $\mathbb{R}_{\mathcal{T}}^{\text{reduced}}$ denote all legal release sequences for $\mathcal{T}$, wherein each job of τ_i, $i = 1, \ldots, k-1$, interferes with a lower priority job, as defined by (11). τ_k is schedulable for each legal R, satisfying (1), iff τ_k is schedulable for each $R' \in \mathbb{R}_{\mathcal{T}}^{\text{reduced}}$.*

We next apply Corollary 1 to speed-up Algorithm 1. Let us extend state definition (7) at time t to

$$\Big(c_i, d_i, p_i, b_i\Big)_{i=1}^{k},$$

where b_i is boolean, such that $b_i(t) = 1$ iff τ_i has a pending job at t, and that job has interfered with a lower priority one by time t inclusive (meaning that condition (11) holds for that τ_i job at some time $t^* \le t$).

(a) A legal release sequence R for $\mathcal{T}$

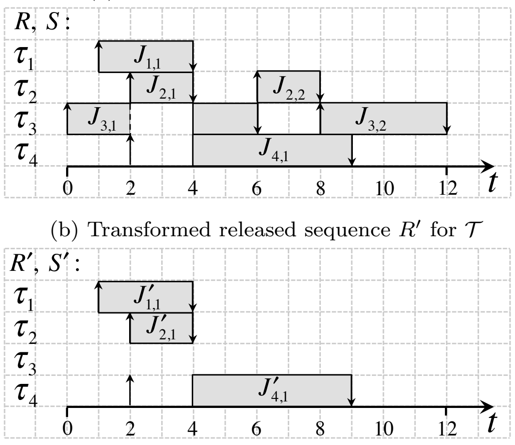


Figure 3: Schedule transformation for Theorem 1

The transition law (8) is extended for $b_i(t)$ accordingly. An initial state is $(0,0,0,0)_{i=1}^{k}$ with $b_i = 0$. For state $(c_i, d_i, p_i, b_i)_{i=1}^{k}$ at time t, b_i is computed by

$$b_i = \begin{cases} 1, & \text{if } s_i(t) = 1 \wedge (\exists \ell > i : q_\ell(t) = 1 \wedge s_\ell(t) = 0) \\ 1, & \text{if } b_i^{\mathsf{prec}} = 1 \ \wedge\ q_i(t) = 1 \ \wedge\ r_i(t) = 0 \\ 0, & \text{otherwise} \end{cases}, \tag{13}$$

where b_i^{prec} corresponds to the preceding state. In the definition above, $b_i = 1$ iff τ_i interferes with a lower priority job at time t (that is the first condition), or τ_i job, pending at time t, has interfered with a lower priority job prior to time t (that is the second condition).

We can determine if a job is non-interfering only after analyzing its entire execution. Therefore, we can check if (11) holds only when job completes in a state transition graph. Below we update the transition law (8) accordingly, by excluding from the analysis every state with jobs that violate (11).

Suppose that state $(c_i, d_i, p_i, b_i)_{i=1}^{k}$ at time t is such that

$$\exists \ell < k : \quad c_\ell = 1 \ \wedge\ s_\ell = 1 \ \wedge\ b_\ell = 0,$$

with s_ℓ computed by (3). Due to Corollary 1, we can safely discard a schedule with such a state from the analysis, because condition (11) is violated for τ_ℓ: $c_\ell > 0$ means that τ_ℓ has a pending job at time t, $b_\ell = 0$ means that τ_ℓ job does not interfere with any lower priority job by time $t+1$ inclusive, and $c_\ell = 1 \wedge s_\ell = 1$ means that τ_ℓ job will be completed by time $t+1$.

Then, the transition law (8) for state g is optimized by adding a pruning constraint

$$\forall g' \in G', \ \ \forall i < k : \quad c'_i = 1 \ \wedge\ s'_i = 1 \ \longrightarrow\ b'_i = 1, \tag{14}$$

where $g' = (c'_i, d'_i, p'_i, b'_i)$ is a successor for g, with G' defined by (8).

Recall a release scenario R as depicted in Fig. 3a. As constraint (14) is violated for job $J_{3,1}$ at time $t = 5$, such R is not considered further for $t \ge 5$. Instead, the test will consider a release scenario, when another job is released by τ_1 or τ_2 at time $t = 5$ (if such a release is feasible), so that $J_{3,1}$ will cause interference on $J_{4,1}$.

Another example is depicted in Fig. 4, showing a reduced state graph, thanks to (14), for $\mathcal{T}$ with parameters as given in Table 1, scheduled upon $m = 2$ processors.

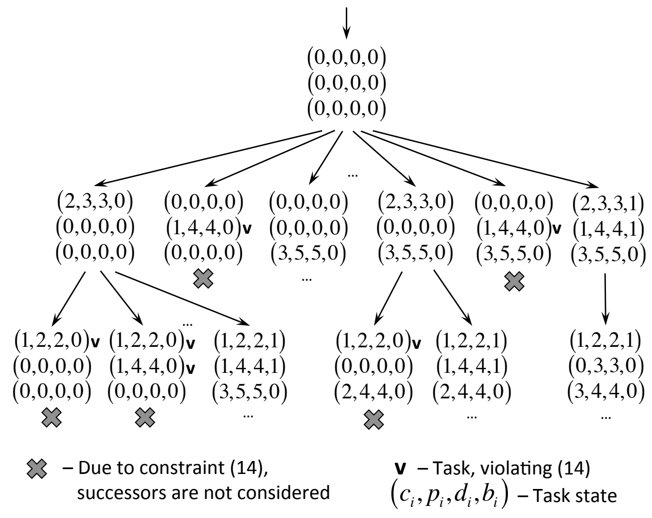


Figure 4: Pruned state transition graph by (14)

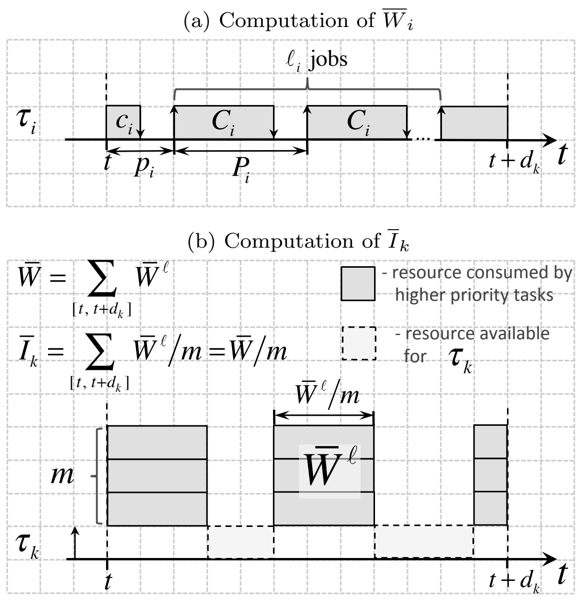

(a) Computation of $\overline{W}_i$
ℓ_i jobs
τ_i
c_i
C_i
C_i
t
p_i
P_i
$t+d_k$
t
(b) Computation of $\bar{I}_k$
$\overline{W} = \sum_{[t,\,t+d_k]} \overline{W}^\ell$
$\bar{I}_k = \sum_{[t,\,t+d_k]} \overline{W}^\ell/m = \overline{W}/m$
- resource consumed by higher priority tasks
- resource available for τ_k
$\overline{W}^\ell/m$
$\overline{W}^\ell$
m
τ_k
t
$t+d_k$
t

Figure 5: Maximum job interference

Constraint (14) allows to totally eliminate the existence of all non-interfering jobs from release scenarios. Later, in Section 5, we report the evaluation results, which confirm the high efficiency of (14) in pruning the state space; that efficiency significantly increases with the number of tasks n.

4.2 Sufficient schedulability condition

Next, we prune the state space G' for Algorithm 1 through applying a sufficient schedulability condition. In fact, if a sufficient schedulability condition holds for some system state, then this state cannot lead to a failure state, and thus we do not need to examine its successors.

At time t, let state $(c_i, d_i, p_i)_{i=1}^k$ be such that $c_k > 0$, meaning that τ_k has a pending job at time t, with remaining execution time c_k and a deadline at time $t + d_k$.

Applying Baruah's analysis [10], the amount of resource allocated to τ_k over time $[t, t + d_k)$ is at least

$$d_k - \bar{I}_k, \tag{15}$$

where $\bar{I}_k$ is the upper bound on the interference, caused by $\tau_1, \ldots, \tau_{k-1}$ on τ_k, computed by (see Fig. 5 for intuition)

$$\bar{I}_k = \frac{\overline{W}}{m}, \tag{16}$$

with the maximum aggregated workload $\overline{W}$ for $\tau_1, \ldots, \tau_{k-1}$ computed by

$$\begin{aligned} \overline{W} &= \sum_{i=1}^{k-1} \overline{W}_i \\ \overline{W}_i &= \min(c_i, d_k) + \ell_i\, C_i + \min(C_i, \Delta_i), \end{aligned} \tag{17}$$

with

$$\ell_i = \max\left(0, \left\lfloor \frac{d_k - p_i}{P_i} \right\rfloor\right) \quad \text{and} \quad \Delta_i = d_k - p_i - \ell_i\, P_i. \tag{18}$$

If the lower bound (15) on the supply allocated to τ_k is not lower than the remaining demand c_k of τ_k, that is $d_k - \bar{I}_k \geq c_k$, then τ_k is guaranteed schedulable. Thus, Algorithm 1 needs to check the successor states for $(c_i, d_i, p_i)_{i=1}^k$ only if

$$d_k - \bar{I}_k < c_k. \tag{19}$$

We have chosen condition (19) due to its low computation time. However, many other tests could be used instead of (19), and their performance remains to be analyzed.

4.3 Critical release instant

Davis and Burns [11] have shown that the worst-case execution scenario for a job of task τ_k occurs when that job is released at such time t ($r_k(t) = 1$), when all m processors are occupied by higher priority tasks $\tau_1, \ldots, \tau_{k-1}$ (that is, $\sum_{\ell=1}^{k-1} q_\ell(t) \geq m$), but there is at least one processor idle during the preceding time interval $[t-1, t)$ (that is, $\sum_{\ell=1}^{k} q_\ell(t-1) < m$):

$$r_k(t) = 1 \quad \Rightarrow \quad \sum_{\ell=1}^{k-1} q_\ell(t) \geq m \;\wedge\; \sum_{\ell=1}^{k} q_\ell(t-1) < m, \tag{20}$$

with $q_\ell(t)$ defined by (2), and $q_\ell(t) = 0$ extended for $t < 0$. Further details can be found in Theorem 1 of [11].

We next adapt such an approach to restrict the release times for $\tau_1, \ldots, \tau_{k-1}$. We first provide an example. Consider $\mathcal{T} = \{\tau_1, \ldots, \tau_4\}$ with parameters as reported in Table 2, scheduled upon $m = 2$ processors. We analyze the schedulability of τ_4.

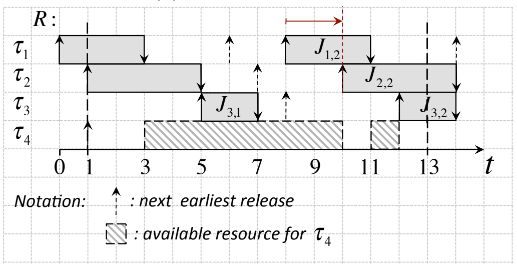
Table 2: Critical release instant: Task set example

i	C_i	P_i	D_i
1	3	6	6
2	4	6	6
3	2	3	3
4	-	12	12

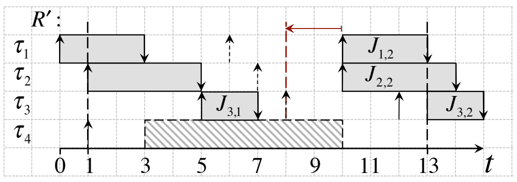
Fig. 6a depicts a legal release sequence for $\mathcal{T}$, denoted by R. In such R, job $J_{1,2}$ of τ_1 causes no interference to other jobs until time 10. Observe also that, after releasing $J_{1,2}$ at time 8, τ_1 cannot release another job until τ_4's deadline at time 13. Then, we can safely postpone the release of $J_{1,2}$ until time 10, as depicted in Fig. 6b.

The same reasoning does not apply, however, to job $J_{3,1}$ of τ_3: delaying the release of $J_{3,1}$ might potentially affect the release time of $J_{3,2}$ (due to the constraint on the minimal

(a) Release sequence R



(b) Release sequence R'



(c) Release sequence R''

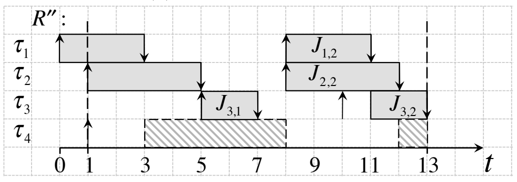


Figure 6: Critical release instants

time separation P_3), and $J_{3,2}$ in turn affects the schedulability of τ_4.

We next formalize the argument. For an arbitrary release sequence R, let time t be such that:

1. τ_k has a pending job at time t: $q_k(t) = 1$;
2. τ_i, with $i < k$, releases a job at time t: $r_i(t) = 1$;
3. τ_i cannot release another job until τ_k's deadline:

$$P_i - D_k + (t - t_r) \geq 0, \tag{21}$$

where t_r denotes the release time for τ_k job, pending at time t.

At such time t, there should be at least $m+1$ pending jobs for $\tau_1, \ldots, \tau_k$:

$$\forall t, i < k: \begin{cases} q_k(t) = 1 \\ r_i(t) = 1 \\ P_i - D_k + t - t_r \geq 0 \end{cases} \Rightarrow \sum_{\ell=1}^{k} r_\ell(t) > m. \tag{22}$$

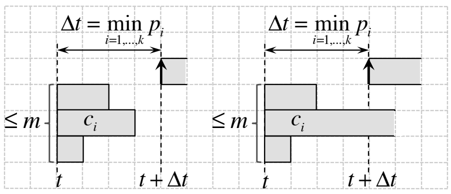
The efficiency of (22) in pruning the state space is higher for lower ratios $P_k/P_{\min}$, with $P_{\min} = \min_{i=1,\ldots,k-1} P_i$; that is due to the presence of (21) in (22).

The release times for $\tau_1, \ldots, \tau_{k-1}$ can be further constrained, by exploring their periods $P_1, \ldots, P_{k-1}$. Recall the release sequence R' as depicted in Fig. 6b. At time 8, no job is released by $\tau_1, \ldots, \tau_3$, although each of them could; the next job is only released 2 time units later, at time 10. To maximize interference on τ_4, we can transform R' into a new R'', by shifting 2 time units left all release instants for $\tau_1, \ldots, \tau_3$, occurred after time 8, as depicted in Fig. 6c. Clearly, interference on τ_k cannot decrease due to such a transformation.

Thus, we require that each R satisfies the condition

$$\sum_{i=1}^{k-1} p_i(t) > 0, \qquad \forall t \tag{23}$$

(a) Case 1



(b) Case 2

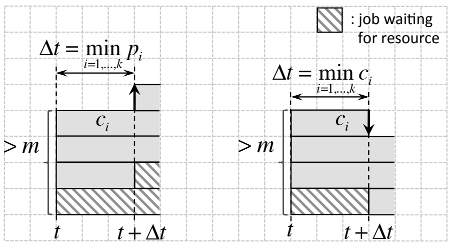


Figure 7: Clock increment Δt: cases

in order to exclude case $\sum_{i=1}^{k-1} p_i(t) = 0$, when each $\tau_1, \ldots, \tau_{k-1}$ can release another job, but none of them does.

Due to lack of space, we do not provide a formal proof for constraints (22) and (23). Such a proof can be conducted by analogy to the proof of Theorem 1 in [11], which results in (20).

We conclude that the set G' of successor states, defined by transition law (8), can be pruned further, by adding constraints (20), (22), and (23).

4.4 State transition

To avoid unnecessary checks for deadline misses at every time instant, we optimize the clock transition between checked system states as follows.

Let state $(c_i, d_i, p_i)_{i=1}^k$, at time t, be such that τ_k does not miss any deadline, i.e., $c_k \leq d_k$ by condition (10).

Suppose first that at most m jobs are pending at $(c_i, d_i, p_i)_{i=1}^k$; that is, $\sum_{i=1}^k \min(c_i, 1) \leq m$ (see Fig. 7a). In this case, no deadline miss can occur until the time when another job can be released, that is time $t + \Delta t$, with Δt defined by

$$\Delta t = \left(\min_{i=1,\ldots,k} p_i \right)_1, \tag{24}$$

where $(x)_1$ denotes $\max(1, x)$.

Suppose instead that more than m jobs are pending at $(c_i, d_i, p_i)_{i=1}^k$ (see Fig. 7b). Let Δt denote the remaining time until the next system event might occur for some task τ_i (job release, completion or deadline):

$$\Delta t = \left(\min_{i=1,\ldots,k} (c_i, p_i, d_i) \right)_1. \tag{25}$$

Let $q' = (c'_i, d'_i, p'_i)_{i=1}^k$ be a successor state for $q = (c_i, d_i, p_i)_{i=1}^k$, Δt time units later, with Δt defined as above. If any of the intermediate states between q and q' is a failure state, then q' is a failure state as well.

Thus, the optimized clock transition Δt between checked system states is computed by (24) and (25):

$$\Delta t = \begin{cases} \left(\min_{i=1,\ldots,k} p_i \right)_1, & \text{if } \sum_{i=1}^k \min(c_i, 1) \leq m \\ \left(\min_{i=1,\ldots,k} (c_i, p_i, d_i) \right)_1, & \text{otherwise.} \end{cases} \tag{26}$$

The transition law (8) is updated accordingly, by replacing the clock increment "1" in (8) and (9) by Δt:

$$(c_i', d_i', p_i')_{i=1}^k \in G' \iff$$

$$\begin{cases} c_i' = (c_i - s_i(t)\,\Delta t)_0 + r_i(t+\Delta t)\,C_i \\ d_i' = (d_i - \Delta t)_0 + r_i(t+\Delta t)\,D_i \qquad i = 1,\dots,k \\ p_i' = (p_i - \Delta t)_0 + r_i(t+\Delta t)\,P_i \end{cases} \tag{27}$$

where $g' = (c_i', d_i', p_i')$. Δt and $s_i(t)$ are defined by (26), (3), and $r_i(t+\Delta t)$ satisfies (1):

$$\begin{aligned} p_i - \Delta t > 0 \quad &\Rightarrow \quad r_i(t+\Delta t) = 0 \\ p_i - \Delta t = 0 \quad &\Rightarrow \quad r_i(t+\Delta t) \in \{0,1\} \end{aligned}$$

4.5 Procedure for the schedulability test

The optimized procedure for an exact schedulability test is as follows. In Algorithm 1, we replace the state transition law (8) by (27), and we incorporate the pruning constraints (14), (19), (20), (22), (23) into (27).

5. EVALUATION

We finally evaluate the performance of the exact schedulability test presented in Section 4.5. The test is implemented using C++ libraries, by extending Bonifaci's tool [5], and its code is publicly available[3].

The experiments are conducted on a hardware platform with the following specifications:

- Processor: Intel Core i7-4710MQ CPU @ 2.5GHz
- Operating memory (RAM): 15,50 GB 1600 MHz
- System type: 64-bit
- Operating system: Ubuntu 14.04 LTS

5.1 Task set generation

Sporadic task sets $\mathcal{T} = \{\tau_i = (C_i, P_i)\}$ with implicit deadlines $D_i = P_i$ are randomly generated by specifying the number of tasks n, the total task set utilization $U_\mathcal{T}$, the maximum individual task utilization $U_{\max}$, and the range for task periods $[P_{\min}, P_{\max}]$.

The minimum period $P_{\min}$ is randomly taken from the range $[3;10]$, and all the task periods are generated such that the specified ratio $P_{\max}/P_{\min}$ holds. Task execution times C_i are chosen by solving the following linear integer optimization problem, using CPLEX:

$$\text{minimize} \quad \left| U_\mathcal{T} - \sum_{i=1}^{n} C_i/P_i \right| + |U_{\max} - C_{i^*}/P_{i^*}|$$

subject to

$$\begin{aligned} &0 < C_i < P_i, \qquad i = 1,\dots,n \\ &\left| U_\mathcal{T} - \sum_{i=1}^{n} C_i/P_i \right| \le \delta_{U_\mathcal{T}} U_\mathcal{T} \\ &|U_{\max} - C_{i^*}/P_{i^*}| \le \delta_{U_{\max}} U_{\max} \\ &C_i/P_i \le C_{i^*}/P_{i^*}, \qquad i = 1,\dots,n, \end{aligned}$$

where C_i, $i = 1,\dots,n$, are integer optimization variables, $|x|$ denotes the absolute value of x, and index i^* corresponds to task τ_{i^*} having the maximum utilization $U_{\max}$; i^* is randomly taken from range $[1,\dots,n]$. We allow a relative deviation $\delta_{U_\mathcal{T}} = 1.5\%$ between the specified value U and the actual tasks utilization $\sum_{i=1}^{n} C_i/P_i$, as well as deviation $\delta_{U_{\max}} = 2.5\%$ between the specified value $U_{\max}$ and the actual maximum task utilization $\max_{i=1,\dots,n} C_i/P_i$.

We randomly generate task sets for the settings reported in Table 3. In each experiment, one key parameter varies, while the rest are left equal to the default values in Table 3.

The choice for parameter values is constrained by a hardware limitation of 16 GB of operating memory. $U_\mathcal{T}$ is chosen such that the pessimism of Guan's test [4] is maximized (that is the case when the usage of an exact test makes more sense). For a thorough evaluation, the pessimism of Guan's test is analyzed for varying $U_\mathcal{T}$ as well.

5.2 Runtime reduction

We first compare the runtime of our test against Bonifaci's test [5][4], for $m = 2$. Figs. 8a, 8b report the average runtime for a varying number of tasks n, and a varying ratio $P_{\max}/P_{\min}$ of task periods, considering only schedulable task sets. We confirm a significant runtime reduction for our test, allowing task sets with larger n and $P_{\max}/P_{\min}$. However, the runtime complexity remains exponential in n.

The second experiment is conducted for $m = 3$. The

Table 3: Key parameters: default values

Settings		
Number of processors, m	2	3
Number of tasks, n	5	7
Task set utilization, $U_\mathcal{T}$	1.6	2.2
Maximum individual task utilization, $U_{\max}$	0.6	0.6
Minimum task period, $P_{\min}$	[3, 10]	[3, 10]
Ratio between the maximum and minimum task periods, $P_{\max}/P_{\min}$	4	4

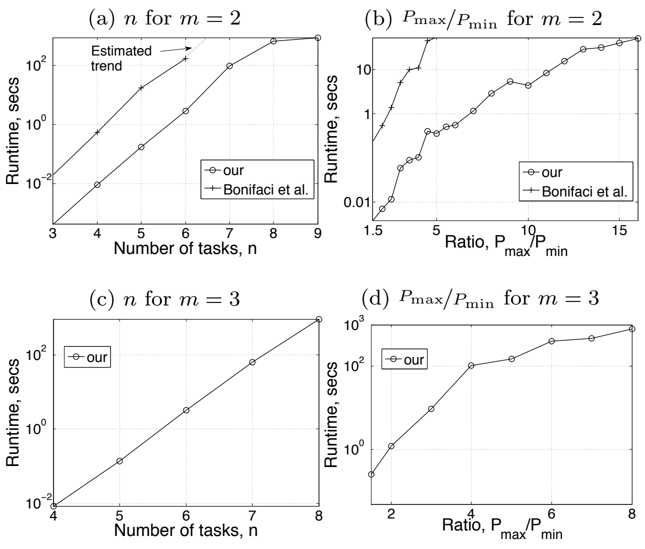


Figure 8: Runtime of exact tests (logarithmic scale)

[3]www.cister.isep.ipp.pt/docs/CISTER-TR-150503

[4]We speeded-up the original code available at http://www.iasi.cnr.it/~vbonifaci/software.php by a factor 10–20 times, by recompiling it at optimized settings.

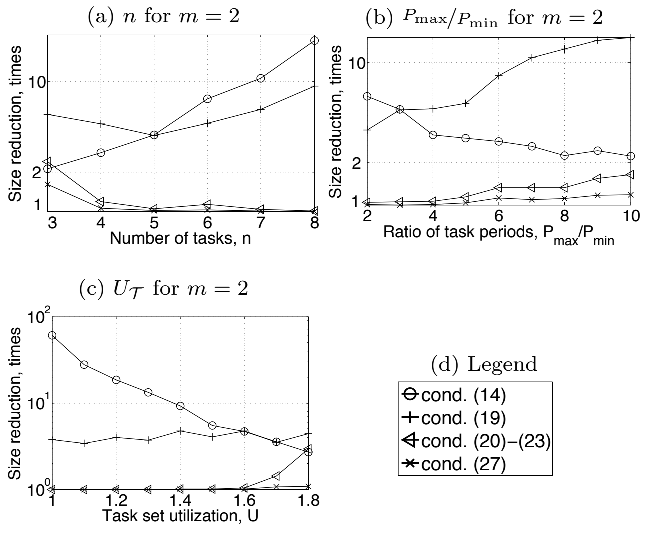


Figure 9: Contribution of each pruning constraint into the state space reduction (logarithmic scale)

average runtime for our test is reported in Figs. 8c, 8d. For such settings, Bonifaci's test requires more than 16 GB of memory in most cases, so that the comparison to our test is infeasible. In 5% of cases, our test exceeds 16 GB as well, and those cases are discarded.

Figs. 9a-9c report the contribution of each pruning constraint into the state space reduction. The plotted size reduction is computed by

$$\mathsf{reduction}_{(x)} = \frac{N_{(x)\text{ excluded}}}{N},$$

where N is the number of states checked by the algorithm employing all pruning constraints (14), (19), (20)-(23), (27), and $N_{(x)\text{ excluded}}$ is the number of states checked by the same algorithm excluding the pruning constraint (x).

Despite of the polynomial space complexity, our evaluation shows that the required system memory for our algorithm (as for all other existing exact tests) increases significantly with the number of tasks and their periods. For example, for task sets comprised of just 10 tasks, and their periods not exceeding 40, the required memory already exceeds 16 GB in most cases.

5.3 Comparison of exact and sufficient tests

To motivate the need for an exact test, we also evaluate the pessimism of Guan's sufficient test [4], which in turn outperforms most of other existing sufficient tests[5]. The experiments are conducted when varying n and $U_{\mathcal{T}}$, for both $m = 2$ and $m = 3$. The results are reported in Figs. 10a-10d. We confirm a significant pessimism of Guan's test, exceeding 50% under certain settings. However, our evaluation is limited to very small task sets, due to the high memory consumption of the exact test.

6. CONCLUSION

We evaluated the performance of sufficient multiprocessor schedulability tests for GFP, and confirmed their significant pessimism. Then, to overcome the major drawbacks of these existing exact tests, which is high computation time and memory consumption, we derived an improved exact test, by extending Bonifaci and Marchetti-Spacamella's work [5]. Evaluation confirmed high efficiency of proposed improvements, although the algorithm remains exponential in time and polynomial in space.

[5]We have implemented Guan's test following an optimized procedure in [12]

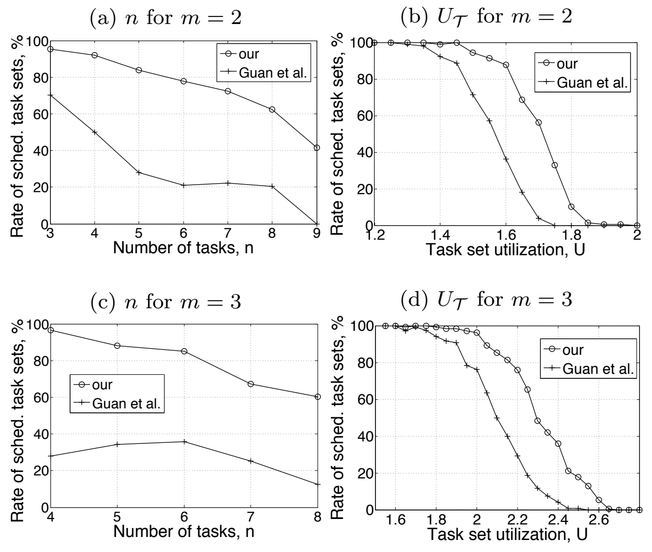


Figure 10: Performance of the sufficient test of Guan *et al.* [4]

7. ACKNOWLEDGMENTS

This work was partially supported by FCT/MEC (Portuguese Foundation for Science and Technology) and cofinanced by ERDF (European Regional Development Fund) under PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER); also by FCT/MEC and the EU ARTEMIS JU within projects ARTEMIS/0003/2012 - JU grant 333053 (CONCERTO) and ARTEMIS/0001/2013 - JU grant 621429 (EMC2); also by FCT/MEC and ERDF through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project FCOMP-01-0124-FEDER-020536 (SMARTS).

8. REFERENCES

[1] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment." *Journal of the ACM (JACM)*, vol. 20, January 1973.

[2] M. Joseph and P. K. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, Oct. 1986.

[3] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1462–1473, Nov. 2004.

[4] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *RTSS*, 2009.

[5] V. Bonifaci and A. Marchetti-Spaccamela, "Feasibility analysis of sporadic real-time multiprocessor task systems," *Algorithmica*, 2012.

[6] T. Baker and M. Cirinei, "Brute-force determination of multiprocessor schedulability for sets of sporadic

hard-deadline tasks," *Principles of Distributed Systems, Lecture Notes in Computer Science*, 2007.

[7] G. Geeraerts, J. Goossens, and M. Lindström, "Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm," *Real-Time Systems*, 2013.

[8] Y. Sun and G. Lipari, "A weak simulation relation for real-time schedulability analysis of global fixed priority scheduling using linear hybrid automata," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, 2014.

[9] N. Guan, Z. Gu, Q. Deng, S. Gao, and G. Yu, "Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking," *Springer Lecture Notes in Computer Science*, vol. 4761, pp. 263–272, 2007.

[10] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2007.

[11] R. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, 2011.

[12] ——, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, 2011.

APPENDIX

The proof of Theorem 1 relies on the following lemma.

LEMMA 1. *Let R, R' be release sequences, as defined in Theorem 1. Let S, S' be resource schedules for R, R' respectively, defined by (3).*

S and S' have the property:

$$\forall i, t: \; q'_i(t) = 1 \quad \longrightarrow \quad s_i(t) = s'_i(t), \tag{28}$$

where $q'_i(t)$ is defined by (2), and $s_i(t)$, $s'_i(t)$ are defined by (3), for S and S' respectively.

Observe that Lemma 1 does not require feasibility of $\mathcal{T}$.

To proof Lemma 1, we explore definition (3) of supply function $s_i(t)$: τ_i job gets supply over time $[t, t+1)$ (that is $q_i(t) = 1 \wedge s_i(t) = 1$), iff the number of higher priority jobs at t is less than m (that is $\sum_{\ell=1}^{i-1} q_\ell(t) < m$):

$$s_i(t) = 1 \quad \Leftrightarrow \quad \begin{cases} q_i(t) = 1 \\ \sum_{\ell=1}^{i-1} q_\ell(t) < m \end{cases} \tag{29}$$

$$s_i(t) = 0 \quad \Leftrightarrow \quad \left[\begin{array}{l} q_i(t) = 0 \\ \begin{cases} q_i(t) = 1 \\ \sum_{\ell=1}^{i-1} q_\ell(t) \geq m \end{cases} \end{array} \right. , \tag{30}$$

$$\forall i, \; t$$

where square brace, "[", denotes logical OR.

Also, the necessary constraints for values of $s_i(t)$ are:

$$s_i(t) = 1 \quad \Rightarrow \quad \sum_{\ell=1}^{i-1} q_\ell(t) < m, \tag{31}$$

$$s_i(t) = 0 \; \wedge \; q_i(t) = 1 \quad \Rightarrow \quad \sum_{\ell=1}^{i-1} q_\ell(t) \geq m, \tag{32}$$

$$\forall i, \; t$$

We are now ready to prove the lemma.

PROOF. We prove the lemma by contradiction. Suppose that (28) is violated for some task τ_{i^*} at time t^*, that is

$$\exists i^*, t^*: \quad q'_{i^*}(t^*) = 1 \; \wedge \; s_{i^*}(t^*) \neq s'_{i^*}(t^*). \tag{33}$$

Without loss of generality, let t^* be the earliest time, when (28) is violated, meaning that

$$\forall i, \; t < t^*: q'_i(t) = 1 \quad \rightarrow \quad s_i(t) = s'_i(t). \tag{34}$$

As $s_i(t)$, $s'_i(t)$ are boolean, (33) might hold in two cases:

$$\begin{cases} s_{i^*}(t^*) = 1 \\ s'_{i^*}(t^*) = 0 \\ q'_{i^*}(t^*) = 1 \end{cases} \quad (35) \qquad \text{or} \qquad \begin{cases} s_{i^*}(t^*) = 0 \\ s'_{i^*}(t^*) = 1 \\ q'_{i^*}(t^*) = 1 \end{cases} \quad (36)$$

We next show that both cases are infeasible, meaning that (33) cannot hold.

Case 1: Substituting (31), (32) into (35), we get that

$$\begin{cases} s_{i^*}(t^*) = 1 \\ s'_{i^*}(t^*) = 0 \\ q'_{i^*}(t^*) = 1 \end{cases} \overset{(31)}{\Rightarrow} \begin{cases} \sum_{\ell=1}^{i^*-1} q_\ell(t^*) < m \\ s'_{i^*}(t^*) = 0 \\ q'_{i^*}(t^*) = 1 \end{cases} \overset{(32)}{\Rightarrow} \begin{cases} \sum_{\ell=1}^{i^*-1} q_\ell(t^*) < m \\ \sum_{\ell=1}^{i^*-1} q'_\ell(t^*) \geq m \end{cases},$$

meaning that exists such a task $\tau_{i^{**}}$, with $i^{**} < i^*$, that

$$q'_{i^{**}}(t^*) = 1 \tag{37}$$

$$q_{i^{**}}(t^*) = 0. \tag{38}$$

From (37), $\tau_{i^{**}}$ has a job at time t^*, in S'. Let J'_{**} denote that job, and let $t'_{r^{**}}$ denote its release time, such that

$$r'_{i^{**}}(t'_{r^{**}}) = 1. \tag{39}$$

As J'_{**} has not been completed by time t^*, J'_{**} has not received C_i units of resource by time t^*:

$$\sum_{t=t'_{r^{**}}}^{t^*-1} s'_{i^{**}}(t) < C_{i^{**}}. \tag{40}$$

From (12), whenever $\tau_{i^{**}}$ releases a job in R', it also releases a job in R. Due to (39) and (12),

$$r'_{i^{**}}(t'_{r^{**}}) = 1 \quad \overset{(12)}{\Longrightarrow} \quad r_{i^{**}}(t'_{r^{**}}) = 1,$$

that is $\tau_{i^{**}}$ releases a job at time $t'_{r^{**}}$, in R. Let J_{**} denote that job, as well as $t_{c^{**}}$ - the completion time for J_{**}.

Considering (38), τ_i has no job pending at time $t^* \geq t'_{r^{**}}$ in S, meaning that J_{**} has been completed by time t^*:

$$t_{c^{**}} \leq t^*$$

$$\sum_{t=t'_{r^{**}}}^{t_{c^{**}}-1} s_{i^{**}}(t) = C_i.$$

what together with (40) yields contradiction to (34), as

$$\sum_{t=t'_{r^{**}}}^{t_{c^{**}}-1} s_{i^{**}}(t) = C_i \quad \bigwedge \quad \sum_{t=t'_{r^{**}}}^{t_{c^{**}}-1} s'_{i^{**}}(t) < C_i$$

implies that

$$\exists t \in [t'_{r^{**}}, t^*) : \ q'_{i^{**}}(t) = 1 \ \wedge \ s_{i^{**}}(t) \neq s'_{i^{**}}(t) \tag{41}$$

Thus, Eq. (35) is infeasible.

Case 2: Suppose that (36) holds. Substituting (29), (30) into (36), we get that

$$\begin{cases} s_{i^*}(t^*) = 0 \\ s'_{i^*}(t^*) = 1 \\ q'_{i^*}(t^*) = 1 \end{cases} \overset{(30)}{\Longleftrightarrow} \begin{cases} \left[\begin{array}{l} q_{i^*}(t^*) = 0 \\ \begin{cases} q_{i^*}(t^*) = 1 \\ \sum_{\ell=1}^{i^*-1} q_\ell(t^*) \geq m \end{cases} \end{array}\right. \\ s'_{i^*}(t^*) = 1 \\ q'_{i^*}(t^*) = 1 \end{cases} \overset{(29)}{\Longleftrightarrow}$$

$$\begin{cases} \left[\begin{array}{l} q_{i^*}(t^*) = 0 \\ \begin{cases} q_{i^*}(t^*) = 1 \\ \sum_{\ell=1}^{i^*-1} q_\ell(t^*) \geq m \end{cases} \end{array}\right. \\ q'_{i^*}(t^*) = 1 \\ \sum_{\ell=1}^{i^*-1} q'_\ell(t^*) < m \end{cases} \Longrightarrow \left[\begin{array}{l} \begin{cases} q_{i^*}(t^*) = 0 \\ q'_{i^*}(t^*) = 1 \end{cases} \\ \begin{cases} q_{i^*}(t^*) = 1 \\ \sum_{\ell=1}^{i^*-1} q_\ell(t^*) \geq m \\ \sum_{\ell=1}^{i^*-1} q'_\ell(t^*) < m \end{cases} \end{array}\right.$$

The first case

$$q_{i^*}(t^*) = 0$$
$$q'_{i^*}(t^*) = 1$$

is infeasible; the proof is conducted by analogy to Case 1, for Eq. (37), (38).

Suppose that the second case holds:

$$q_{i^*}(t^*) = 1 \tag{42}$$

$$\sum_{\ell=1}^{i^*-1} q_\ell(t^*) \geq m \tag{43}$$

$$\sum_{\ell=1}^{i^*-1} q'_\ell(t^*) < m \tag{44}$$

Due to (43) and (44), there exists such a task $\tau_{i^{**}}$, with $i^{**} < i^*$, that satisfies the constraints:

$$q_{i^{**}}(t^*) = 1 \tag{45}$$

$$\sum_{\ell=1}^{i^{**}} q_\ell(t^*) = m \tag{46}$$

$$q'_{i^{**}}(t^*) = 0. \tag{47}$$

Let J_{**} denote $\tau_{i^{**}}$ job, pending at time t^* in S (there is such a job, due to (45)), and let $t_{r^{**}}$ denote its release time:

$$r_{i^{**}}(t_{r^{**}}) = 1,$$

$$\sum_{t=t_{r^{**}}}^{t^*} s_{i^{**}}(t) < C_i. \tag{48}$$

Observe that J_{**} interferes with a lower priority job at time t^*: a processor is assigned to J_{**} at time t^* (that is due to (45) and (46)), and the number of pending jobs at t^* exceeds m (that is due to (42) and (43)). From definition (12) of R':

$$\begin{cases} r_{i^{**}}(t_{r^{**}}) = 1 \\ \text{(11) holds for } J_{**} \end{cases} \overset{(12)}{\Longrightarrow} \quad r'_{i^{**}}(t_{r^{**}}) = 1,$$

meaning that $\tau_{i^{**}}$ released a job at time $t_{r^{**}}$, in R'. Let J'_{**} denote that job, and $t'_{c^{**}}$ - its completion time. Due to (47), $\tau_{i^{**}}$ has no pending job at t^*, in S', meaning that J'_{**} has been completed by time t^*:

$$t'_{c^{**}} \leq t^*$$

$$\sum_{t=t_{r^{**}}}^{t'_{c^{**}}-1} s'_{i^{**}}(t) = C_i$$

what together with (48) contradicts to assumption (34):

$$\begin{cases} t'_{c^{**}} \leq t^* \\ \sum_{t=t_{r^{**}}}^{t'_{c^{**}}-1} s'_{i^{**}}(t) = C_i \end{cases} \quad \bigwedge \quad \sum_{t=t_{r^{**}}}^{t^*} s_{i^{**}}(t) < C_i$$

$$\sum_{t=t'_{r^{**}}}^{t'_{c^{**}}-1} s'_{i^{**}}(t) = C_i \quad \bigwedge \quad \sum_{t=t'_{r^{**}}}^{t'_{c^{**}}-1} s_{i^{**}}(t) < C_i,$$

meaning that (34) is violated:

$$\exists t \in [t_{r^{**}}, t^*) : \ q_{i^{**}}(t) = 1 \ \wedge \ s_{i^{**}}(t) \neq s'_{i^{**}}(t)$$

We conclude that both cases, (35) and (36), are infeasible, and lemma statement (28) holds always.

□

We finally prove Theorem 1.

PROOF FOR THEOREM 1. *Necessity:* We first prove that feasibility of R implies feasibility of R'. The proof is conducted by contradiction. Suppose that exist such R, R', satisfying (12), that R is feasible, but R' is infeasible. For R', let t_{dm} denote the earliest missed deadline for τ_k; consequently, $t_{\mathsf{dm}} - D_k$ is the release time of the job missing the deadline:

$$r'_k(t_{\mathsf{dm}} - D_k) = 1, \tag{49}$$

$$\sum_{t=t_{\mathsf{dm}}-D_k}^{t_{\mathsf{dm}}-1} s'_k(t) < C_k. \tag{50}$$

Due to (12), if τ_k has released a job at time $(t_{\mathsf{dm}} - D_k)$ in R', then it has also released a job in R:

$$r_k(t_{\mathsf{dm}} - D_k) = 1. \tag{51}$$

Due to the assumption that R is feasible,

$$\sum_{t=t_{\mathsf{dm}}-D_k}^{t_{\mathsf{dm}}-1} s_k(t) = C_k, \tag{52}$$

what contradicts to (50) due to Lemma 1.

Sufficiency: The sufficiency proof shows that feasibility of R' implies feasibility of R. Such a proof is conducted by analogy to the necessity proof, by swapping functions $r_k(t)$, $s_k(t)$ and $r'_k(t)$, $s'_k(t)$ in Eq. (49)-(52).

The theorem follows.

□