Response Time Analysis with Limited Carry-in for Global Earliest Deadline First Scheduling

Youcheng Sun, Giuseppe Lipari

Response Time Analysis with Limited Carry-in for Global Earliest Deadline First Scheduling*

Youcheng Sun
Scuola Superiore Sant'Anna
y.sun@sssup.it

Giuseppe Lipari
Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, Lille, France
giuseppe.Lipari@univ-lille1.fr

Abstract

We address the problem of schedulability analysis for a set of sporadic real-time tasks scheduled by the Global Earliest Deadline First (G-EDF) policy on a multiprocessor platform. State-of-the-art tests for schedulability analysis of multiprocessor global scheduling are often incomparable. That is, a task set that is judged not schedulable by a test may be verified to be schedulable by another test, and vice versa.

In this paper, we first develop a new schedulability test that integrates the limited carry-in technique and Response Time Analysis (RTA) procedure for Global EDF schedulability analysis. Then, we provide an over-approximate variant of this test with better run-time efficiency. Later, we extend these two tests to self-suspending tasks. All schedulability tests proposed in the paper have provable dominance over their state-of-the-art counterparts.

Finally, we conduct extensive comparisons among different schedulability tests. Our new tests show significant improvements for schedulability analysis of Global EDF.

1 Introduction and State of the Art

One important class of real-time schedulers for multiprocessor systems is the class of *global scheduling policies*. According to global scheduling, all ready tasks are enqueued in a single ready queue, and the m highest-priority tasks are executed on the m processors. In preemptive global scheduling, a task executing on one processor may be preempted by a higher-priority task and later resume execution on a different processor: therefore, we say that this class of scheduling policies allows *migrations* of tasks among processors. Experimental comparison

*Submitted to IEEE Real-time Systems Symposium, Dec. 2015

in the Linux operating system shows that global/clustered configurations are a viable solution for multiprocessor platforms [18].

The two most popular global scheduling algorithms are Global Fixed Priority (G-FP) and Global Earliest Deadline First (G-EDF). In G-FP scheduling, a fixed priority is assigned to each task and all jobs of the task share the assigned priority; in G-EDF scheduling, a job's priority is inversely proportional to its associated deadline, a smaller absolute deadline corresponding to a higher priority.

Testing the schedulability of a set of sporadic real-time tasks on a multiprocessor platform with global scheduling is a challenging problem. Unlike in the single processor case, in multiprocessor global scheduling, the worst-case release pattern is unknown. Therefore, any exact schedulability condition for a set of sporadic tasks scheduled by global scheduling requires to test a very high number of release patterns of the tasks.

Baker and Cirinei [3] proposed the first exact analysis by building a finite state machine that represents all possible combinations of arrival times and execution interleavings. Bonifaci et al. [10] studied the feasibility problem of sporadic tasks in multiprocessor by reducing it to a safety game. Geeraerts et al. [13] improved Baker and Cirinei's method by using an anti-chain technique. More specifically, they developed a simulation relation between states of the underlying finite automaton. Sun and Lipari [21] recently developed a similar method using continuous time and Linear Hybrid Automata. However, all these exact solutions suffer from poor scalability due to the exponential number of release patterns.

For this reason, most researchers focused on deriving approximated analyses, i.e. *sufficient* conditions for schedulability. We say that a schedulability test A *dominates* another schedulability test B, denoted as $A \succeq B$, if every task set that is deemed schedulable by B is also deemed schedulable by A, and there may exist sets of tasks which are deemed schedulable by A but not by B.

Baker [1] developed a sophisticated analysis technique based on the concept of *problem window*. The technique consists in checking the schedulability of one task at a time: first, the problem window is selected equal to the interval between the arrival time and the deadline of one instance of the task under analysis; then, the *interference* of higher-priority jobs is computed and taken into account in the schedulability analysis. Interference is divided into *carry-in* jobs (i.e. jobs which may start executing before the problem window and whose computation time only partially contributes to the interference) and non-carry-in jobs (i.e. jobs whose arrival time and execution time are contained in the problem window).

Baker's technique has since been extended by many researchers who tried to improve the estimation of the interfering workload. Bertogna et al. [9] discovered that for each competing task with very high workload in the problem window, the part of its workload that has to be executed in parallel with the analysed task should not be taken into account in the actual interference. Later, Bertogna and Cirinei [8] applied this technique to iterative Response Time Analysis (RTA) of global scheduling.

Another breakthrough was proposed by Baruah [4]. His technique tries to limit the number of carry-in tasks. Although such a technique was originally conceived for G-EDF, Guan et al. [14] combined it with the RTA technique, obtaining more precise schedulability tests for G-FP scheduling. Sun et al. [22] recently improved over the result of [14] at the cost of a higher complexity through explicitly enumerating all possible carry-in tasks. They also restricted the analysis to a subset of release patterns that could lead to the worst-case response time for a task. In case of G-FP schedulability analysis, the following chain of domination holds: [22]$\succeq$[14]$\succeq$[8]. In [19], there is another G-FP test based on limiting carry-in workload, but it is incomparable with the tests just mentioned. Lee and Shin [16] generalised the limited carry-in idea to any work-conserving algorithms.

When it comes to G-EDF, besides the tests in [1, 9, 8, 4], many other tests have been proposed, like [2] and [5], and none of these tests could dominate the others. However, according to empirical evaluations [7], tests in [4] and [8] are shown to have better average performance. In the following work, we refer to the test in [4] by Baruah as Bar and the test in [8] by Bertogna and Cirinei as BC.

In [17] a compositional theory is proposed to improve the overall schedulability results, which explores the sufficient condition such that to apply existing over-approximate tests for a subset of total tasks on a subset of processors. While it is likely that the dominance relation between underlying schedulability tests can be preserved also for the composition result, a further study is out of the scope of this paper. Very recently, a "divide-and-conquer" approach was proposed in [15] and applied to BC, which additionally finds certain schedulable task sets that BC fails to detect.

In certain applications, tasks are allowed to suspend their execution for some time (for example when waiting data from external devices). The naive way to treat suspension time is to consider it as execution time, but it may be pessimistic. Liu and Anderson [19] provided the first suspension-aware schedulability tests for G-FP and G-EDF scheduling. In [19] a task can self-suspend at any phase of its execution. Later, Tong and Liu [23] studied schedulability of self-suspending tasks with specific suspension patterns.

1.1 Contributions of this paper

The objective of this work is to further improve the performance of schedulability tests for G-EDF scheduling on a multiprocessor platform. We develop a new schedulability test for G-EDF that *dominates* both Bar and BC. Also, we provide an over-approximate version of this test with lower run-time complexity, while preserving the dominance over Bar and BC. Then, we extend the new tests to the context of self-suspending tasks. We extensively compared the performance of state-of-the-art tests on a variety of synthetically generated task sets. The results show that the new tests substantially improve existing works for G-EDF scheduling.

The paper is organised as follows. Section 2 introduces the system model.

In Section 3, we recall previous knowledge for G-EDF schedulability analysis, mainly on Bar and BC. In Section 4, we develop new tests for schedulability analysis of G-EDF. In Section 5, these new tests are then adapted for a system with self-suspending tasks. In Section 6, we present extensive simulations to evaluate the performance of our tests with respect to existing ones. Finally, we conclude the work in Section 7.

2 System Model

A sporadic task $\tau_i = (C_i, D_i, T_i)$ is characterised by a Worst-Case Execution Time (WCET) C_i, a relative deadline D_i, and a minimum inter-arrival separation time T_i, which, with an abuse of notation, is sometimes called period. We require that $C_i \leq D_i \leq T_i$. The utilisation of a task is defined as $U_i = \frac{C_i}{T_i}$.

Each task τ_i can release an infinite sequence of *jobs* (also called *instances*); the time interval between two successive job releases is at least T_i. Each job $J_{i,j}$ from task τ_i is characterised by its *release time* $r_{i,j}$ (with $r_{i,j+1} - r_{i,j} \geq T_i, j \geq 0$) and its *absolute deadline* $d_{i,j} = r_{i,j} + D_i$. A released job must finish its execution within its deadline: we denote the finishing time of $J_{i,j}$ as $f_{i,j}$. We say that a job is *active* if the job has been released, but it has not finished its execution. Since we assume $D_i \leq T_i$, this means a task can at most have one active job at any moment. The Worst-Case Response Time (WCRT) of a task is defined as $R_i = \max_j\{f_{i,j} - r_{i,j}\}$ and a task is schedulable if every job completes no later than the corresponding absolute deadline, i.e. if $R_i \leq D_i$. In this work, we consider discrete time domain and all values of $r_{i,j}$, $d_{i,j}$ and $f_{i,j}$ are assumed to be positive integers.

Let $\mathcal{T}$ be a set of n independent sporadic tasks: $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$. The task set total utilisation is $U_{tot} = \sum_{\tau_i \in \mathcal{T}} U_i$. We assume that $\mathcal{T}$ is executed on m ($< n$) identical processors. To avoid extreme cases, U_{tot} is required to be strictly less than m in this work.

In Section 5, we discuss schedulability analysis for *self-suspending* tasks, that is, a task may suspend itself during execution. A self-suspending task τ_i is characterised by (C_i, D_i, T_i, S_i), where S_i denotes that a job of τ_i can at most suspend itself for S_i time units. Note that a job can suspend multiple times as long as the cumulative suspension time is no more than S_i, and it can begin or end with a suspension phase. We require that $C_i + S_i \leq D_i$.

A task that never suspends itself is also called a *computational* task for explicitly distinguishing it from self-suspending tasks. A computational task can be regarded as a special self-suspending task with $S_i = 0$. When there are both self-suspending and computational tasks in $\mathcal{T}$, we use $\mathcal{T}^s$ to denote the subset of self-suspending tasks and $\mathcal{T}^c$ to denote the subset of computational tasks. That is, $\mathcal{T} = \mathcal{T}^s \cup \mathcal{T}^c$ and $\mathcal{T}^s \cap \mathcal{T}^c = \emptyset$.

Tasks in $\mathcal{T}$ are supposed to be scheduled according to a Global Earliest Deadline First (G-EDF) preemptive scheduling policy. A job may execute upon any processor, and a preempted or self-suspended job may later resume execution upon the same processor as, or upon a different processor from, the one it

had been executing.

3 Global EDF Schedulability: Prior Results

In this section, we briefly summarise Bar [4] and BC [8]. To simplify the expressions, we define the following notations: $[\![x]\!]_y = \max\{x, y\}$, $[\![x]\!]^y = \min\{x, y\}$ and $[\![x]\!]_y^z = [\![[\![x]\!]_y]\!]^z$.

3.1 Interference

When analysing the schedulability of a task τ_k under G-EDF scheduling, a problem window is often assumed. A *problem window* is a time interval $[a, b)$ where time point b is coincident with the deadline for some job of τ_k, which we call the *target* job.

(Demand bound function) In EDF scheduling, the concept of *demand bound function* plays an important role for schedulability analysis. For any t, the demand bound function $\mathsf{DBF}_i(t)$ of a task τ_i bounds the maximum cumulative execution requirement by jobs of τ_i that have *both arrival time and deadline within any interval of length* t. It is formally defined as follows.

$$\mathsf{DBF}_i(t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \tag{1}$$

(Interference) In order to check if the target job of τ_k will miss the deadline, the interference from an interfering task τ_i is estimated. The interference $I_{i,k}(t)$ denotes the cumulative length of all sub-intervals of an interval $[a, b)$ of length t such that the target job of τ_k is active but cannot be executed, while τ_i is being executed. Since computing the exact interference is very difficult, we will compute an upper bound. According to the observation in [9], one trivial upper bound is:

$$0 \leq I_{i,k}(t) \leq (t - C_k + 1). \tag{2}$$

For simplicity of notation, in this paper we abuse the term "interference" to also denote any upper bound on the interference. Tasks that interfere with τ_k's execution can be differentiated into two classes: *carry-in* (CI) tasks and *non-carry-in* (NC) tasks. Given a problem window $[a, b)$, a task is called a CI task if it has a job released before the beginning of the problem window and still active at time point a, otherwise it is a NC task. CI tasks can generate more interference in the problem window.

For a NC task τ_i, its interference in a problem window can be bounded by using the DBF function directly:

$$I_i^{NC}(t) = \mathsf{DBF}_i(t).$$

If τ_i is a CI task, its possible interference is estimated as:

$$I_i^{CI}(t) = \left\lfloor \frac{t}{T_i} \right\rfloor \times C_i + [\![(t \bmod T_i) - D_i + R_i]\!]_0^{C_i}.$$

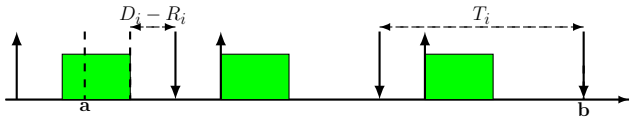


Figure 1: Maximum CI interference under G-EDF

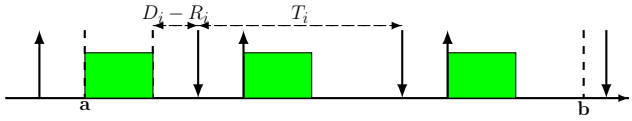


Figure 2: Maximum workload in a problem window

The worst-case CI interference corresponds to the scenario depicted in Figure 1. Note that, when estimating the interference of a task τ_i, it is safe to upper bound R_i with D_i. Among the sequence of jobs from τ_i that may interfere with the target job, the first job is called *carry-in job*, and the last one is called *carry-out job*. The CI task's interference in the problem window is maximised when: 1) deadline of carry-out job is coincident with b, 2) every job of τ_i is released as soon as possible, and 3) the carry-in job exactly finishes the execution with its worst-case response time. The inequality $I_i^{NC}(t) \leq I_i^{CI}(t)$ always holds.

3.2 Bertogna and Cirinei's test

The *workload* of a task over a time interval represents the amount of computation that the task requires in the interval. Bertogna and Cirinei [8] formulated a generic upper bound to the workload of a task τ_i over an arbitrary time interval with some length L that does not depend on the specific scheduling algorithm:

$$\mathfrak{W}_i(L) = N_i(L)C_i + [\![L + R_i - C_i - N_i(L)T_i]\!]_0^{C_i}$$

where $N_i(L) = \left\lfloor \frac{L+R_i-C_i}{T_i} \right\rfloor$.

The situation that brings the worst-case workload is depicted in Figure 2: the carry-in job starts execution at the beginning of the window and finishes exactly with its worst-case response time; and every successive instance of τ_i arrives as soon as possible.

Finally, BC relies on a classic iterative response time analysis.

Theorem 1 (Theorem 6 in [8])**.** *An upper bound on the response time of a task τ_k in a G-EDF scheduled multiprocessor system can be derived by the fixed point iteration over X of the following expression, starting with $X = C_k$.*

$$X \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} I_{i,k}(X) \right\rfloor$$

with $I_{i,k}(X) = \min(\mathfrak{W}_i(X), I_i^{CI}(D_k), X - C_k + 1)$.

For every task, its response time R_i is needed to compute $\mathfrak{W}_i(X)$ and $I_i^{CI}(D_k)$. However, R_i is unknown. This can be solved by the following iterative procedure.

1. For every task τ_i, its R_i is initialised to D_i.
2. Apply Theorem 1 to every task in the system. If the resulting response time is $> D_i$ for some task, then that task is marked as "potentially unschedulable". If there are no potentially unschedulable tasks, the task set is declared to be schedulable.
3. Repeat step (2) until there is no R_i update.

Please note that in BC all tasks are considered as CI tasks.

3.3 Baruah's test

Baruah [4] derived a sufficient schedulability test for G-EDF. Given the problem window $[a, b)$ with length D_k, the idea in Bar is to extend the interval back to some point s_0 at which at least one of the m processors is idle, and from s_0 to a all processors are busily executing jobs with absolute deadlines no larger than the target one's. Let us define $A_k = a - s_0$. Such a time interval $[s_0, s_0 + A_k)$ is called the **busy period**. Thanks to this extension, there are at most $(m-1)$ CI tasks at time point s_0. In order for τ_k's target job to meet its deadline, it is sufficient that all m processors are simultaneously occupied by higher-priority jobs for no more than $A_k + (D_k - C_k)$ time units over $[s_0, b)$.

Different from BC, Bar explicitly differentiates the interference caused by the NC task ($I_{i,k}^{NC}$) and interference caused by the CI task ($I_{i,k}^{CI}$) in the extended problem window.

$$I_{i,k}^{NC} = \begin{cases} \left[\!\left[I_i^{NC}(A_k + D_k) \right]\!\right]^{A_k + D_k - C_k + 1} & \text{if } i \neq k \\ \left[\!\left[I_i^{NC}(A_k + D_k) - C_k \right]\!\right]^{A_k} & \text{if } i = k \end{cases}$$

$$I_{i,k}^{CI} = \begin{cases} \left[\!\left[I_i^{CI}(A_k + D_k) \right]\!\right]^{A_k + D_k - C_k + 1} & \text{if } i \neq k \\ \left[\!\left[I_i^{CI}(A_k + D_k) - C_k \right]\!\right]^{A_k} & \text{if } i = k \end{cases}$$

Let us define $I_{i,k}^{DIFF} = I_{i,k}^{CI} - I_{i,k}^{NC}$, then the following theorem holds.

Theorem 2 (Theorem 3 in [4])**.** *A task set $\mathcal{T}$ is schedulable under G-EDF if, for any $\tau_k \in \mathcal{T}$ and all $0 \leq A_k \leq \overline{A}_k$,*

$$\Omega \leq m(A_k + D_k - C_k) \tag{3}$$

where Ω is the total interference:

$$\Omega = \left(\sum_{\tau_i \in \mathcal{T}} I_{i,k}^{NC} + \sum_{\textit{the } (m-1) \textit{ largest}} I_{i,k}^{DIFF} \right) \tag{4}$$

and $\overline{A}_k$ *is defined as follows:*

$$\overline{A}_k = \frac{C_\Sigma + D_k U_{tot} - mD_k + \sum_{\tau_i \in \mathcal{T}} (T_i - D_i)U_i + mC_k}{m - U_{tot}} \tag{5}$$

with C_Σ *denoting sum of the* $(m-1)$ *largest WCETs among tasks.*

Additionally, the condition in Equation (4) needs only to be tested on the A_k's values at which $\mathsf{DBF}_i(A_k + D_k)$ changes for some task τ_i. When $m = 1$, Bar is equivalent to the exact schedulability test for EDF in [6].

4 An Improved Schedulability Test for G-EDF

The above presented BC and Bar tests use two different techniques: BC shows us that the classic iterative response time analysis procedure is still effective in multiprocessor schedulability analysis; Bar contributes the idea that we may more precisely estimate the interference on multiprocessor global scheduling by limiting the number of CI tasks. However, all tasks in BC are considered as CI tasks, whereas Bar directly measures the interference in a time interval instead of using the iterative analysis. Therefore, in this paper we combine the two techniques to take advantage of their strong points.

Before proceeding with the presentation of the new algorithm, we quickly recapitulate the related concepts; we further propose the definition of a *sub problem window*, as depicted in Figure 3.

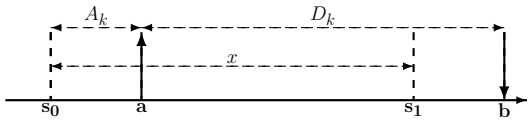


Figure 3: Different problem windows

We are checking the schedulability of a target task τ_k, so we select a *target job* of τ_k in the window $[a, b)$ with a and b corresponding to its release time and deadline, respectively. We still use the extended problem window $[s_0, b)$, where s_0 is the earliest time point before a such that within $[s_0, a)$ all processors are busily executing jobs with absolute deadlines smaller than or equal to the target job's and $A_k = a - s_0$ is the length of this busy period.

In the following, instead of simply computing the interference generated by a task in the extended problem window $[s_0, b)$, we follow the iterative technique in BC to calculate the interference. We define a new time interval $[s_0, s_1)$, called *sub problem window of the extended one*, such that $s_0 < s_1 \leq b$, and we would like to compute the interference in the sub problem window $[s_0, s_1)$. We denote $x = s_1 - s_0$.

4.1 Interference in a sub problem window

Before upper bounding the interference in the sub problem window $[s_0, s_1)$, we first formulate the workload generated by a task τ_i in this sub problem window, denoted as $W_i(x, \mathcal{L})$ with $\mathcal{L} = A_k + D_k$. We use $W_i^{NC}(x, \mathcal{L})$ for NC tasks and $W_i^{CI}(x, \mathcal{L})$ for CI tasks, and $W_i^{DIFF}(x, \mathcal{L}) = W_i^{CI}(x, \mathcal{L}) - W_i^{NC}(x, \mathcal{L})$.

Algorithm 1 calculates the maximum workload produced by a NC task inside $[s_0, s_1)$. It follows the worst-case release pattern for $W_i^{NC}(x, \mathcal{L})$ such that

- the first job (carry-in job) of τ_i is released at time s_0;
- successive jobs of τ_i are released as soon as possible.

Algorithm 1: $W_i^{NC}(x, \mathcal{L})$

```
1 W ← 0, p ← 0
2 while p < x do
3     if p + D_i ≤ L then
4         W ← W + ⟦x − p⟧^{C_i}
5         p ← p + T_i
6     else
7         break
8 return W
```

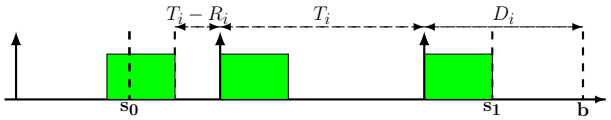


Figure 4: The worst-case arrival pattern for $W_i^{CI}(x, \mathcal{L})$

As for a CI task τ_i, its worst-case release pattern for $W_i^{CI}(x, \mathcal{L})$ corresponds to the scenario in Figure 4:

- the carry-in job finishes its execution with τ_i's worst-case response time;
- successive jobs of τ_i are released as soon as possible;
- the carry-out job arrives as late as possible and finishes its worst-case execution within the sub window.

It is easy to see that any other scenario produces a workload that is not greater than this one. Note that all interfering jobs, including the carry-out one, have their absolute deadlines no later than b.

Algorithm 2 uses this worst-case release pattern to compute $W_i^{CI}(x, \mathcal{L})$. At first, it computes the latest possible arrival time for the carry-out job, denoted

as $[\![x - C_i]\!]^{\mathcal{L}-D_i}$. If the resulting arrival time is less than 0, it means that there is at most one interfering job of τ_i in the (sub) window and it must arrive before the beginning of the busy period; then lines 3-6 in the algorithm deal with such a situation. In the other case, there will be $(N+1)$ interfering jobs of τ_i that can contribute their complete worst-case executions; and the remaining part (lines 7-9) of the algorithm deals with this situation.

Algorithm 2: $W_i^{CI}(x, \mathcal{L})$

```
1 W ← 0
2 p ← [[x − C_i]]^{L−D_i}
3 if p < 0 then
4 |   W ← [[L − (D_i − R_i)]]^{C_i}
5 |   W ← [[W]]_0^x
6 |   return W
7 N ← ⌊p / T_i⌋
8 W ← (N + 1) · C_i + [[p mod T_i − (T_i − R_i)]]_0^{C_i}
9 return W
```

For τ_k, we only need to consider its job releases before the target one. Then, its NC and CI workload can be further bounded.

$$W_k^{NC}(x, \mathcal{L}) = \left[\!\left[W_k^{NC}(x, \mathcal{L})\right]\!\right]^{I_k^{NC}([\![\mathcal{L}-T_k]\!]_0)}$$
$$W_k^{CI}(x, \mathcal{L}) = \left[\!\left[W_k^{CI}(x, \mathcal{L})\right]\!\right]^{I_k^{CI}([\![\mathcal{L}-T_k]\!]_0)}$$

After computing the workload, the interference can be bounded by the property in Equation (2). The corresponding NC interference and CI interference by τ_i on the target job of τ_k are denoted as $I_{i,k}^{NC}(x, A_k)$ and $I_{i,k}^{CI}(x, A_k)$. For any $x \geq A_k + C_k$, the interference from τ_i is bounded as follows.

$$I_{i,k}^{NC}(x, A_k) = \left[\!\left[W_i^{NC}(x, \mathcal{L})\right]\!\right]^{x-C_k+1}$$
$$I_{i,k}^{CI}(x, A_k) = \left[\!\left[W_i^{CI}(x, \mathcal{L})\right]\!\right]^{x-C_k+1}$$

$I_{i,k}^{DIFF}(x, A_k)$ is defined as $(I_{i,k}^{CI}(x, A_k) - I_{i,k}^{NC}(x, A_k))$.

Given the sub window $[s_0, s_1)$, we now demonstrate how to upper bound the total interference on the target job.

- According to the limited carry-in technique in Bar, we can derive an upper bound to the total interference, denoted as Ω_1 such that

$$\Omega_1 = \sum_{\tau_i \in \mathcal{T}} I_{i,k}^{NC}(x, A_k) + \max_{\text{the } m-1 \text{ largest}} I_{i,k}^{DIFF}(x, A_k) \tag{6}$$

- On the other hand, given that $[s_0, a)$ is a busy period with length A_k, we can formulate another upper bound Ω_2 such that

$$\Omega_2 = m \cdot A_k + \sum_{i \neq k} I_{i,k}^{CI}(x - A_k, 0) \tag{7}$$

  where $\sum_{i \neq k} I_{i,k}^{CI}(x - A_k, 0)$ upper bounds the maximum interference in interval $[a, s_1)$ by assuming all tasks different from τ_k as CI tasks.

- Finally, we define the total interference upper bound in the sub window $[s_0, s_1)$ as $\Omega(x, A_k)$, which is the smaller one between Ω_1 and Ω_2; that is

$$\Omega(x, A_k) = [\![\Omega_1]\!]^{\Omega_2} \tag{8}$$

4.2 The new schedulability test

Now, we are going to formulate a new RTA procedure for G-EDF schedulability analysis that integrates the limited carry-in technique, and we call the new test "RTA with Limited Carry-in for multiprocessor global EDF scheduling" (RTA-LC-EDF)[1].

Theorem 3. ***(RTA-LC-EDF)*** *Given an A_k value, let us say X_{A_k} be solution of the following iteration starting from $X = A_k + \phi$ with $\phi = C_k$.*

$$X \leftarrow \phi + \left\lfloor \frac{\Omega(X, A_k)}{m} \right\rfloor \tag{9}$$

Then, the response time upper bound of τ_k is

$$R_k = \max_{\forall A_k} \{X_{A_k} - A_k\}.$$

Proof. The theorem can be proved by showing that, for any A_k such that within the time interval $[s_0, a)$ all processors are fully occupied by higher-priority jobs, X_{A_k} is an upper bound of the target job's finishing time (let us say $s_0 = 0$) subject to this specific A_k value.

Suppose Equation (9) converges to X_{A_k}. If X_{A_k} is not an upper bound of τ_k's finishing time, then there would be $\lfloor \frac{\Omega(X_{A_k}, A_k)}{m} \rfloor + \phi > X_{A_k}$, which contradicts with Equation (9)'s convergence at X_{A_k}. □

Then, the same refinement procedure as in BC can be applied to RTA-LC-EDF.

1. We start by setting $R_i = D_i$ for every task and by marking all tasks as "potentially unschedulable".

[1] This naming convention can be traced back to [11].

2. We compute R_i for every task by using the iterative procedure in Theorem 3. For a potentially unschedulable task, if the resulting response time is $\leq D_i$, then it is marked as "schedulable" and the value of R_i is updated; for a schedulable task, if the resulting response time $< R_i$, then R_i is updated.

3. If no task has R_i update, the iteration stops; otherwise, go back to step (2).

In the following, we are going to prove that the new test dominates Bar and BC: if a task set is decided schedulable by Bar or BC, the same result is returned by RTA-LC-EDF.

Theorem 4. *RTA-LC-EDF $\succeq$ Bar and RTA-LC-EDF $\succeq$ BC.*

Proof. Here we give the key points to prove RTA-LC-EDF's dominance over Bar.

- For any A_k and for any $x \in [A_k + C_k, A_k + D_k]$, there is $\Omega \geq \Omega(x, A_k)$. Remember that Ω (Equation (4)) is the total interference, with respect to corresponding A_k value, computed by Bar in the extended problem window; and $\Omega(x, A_k)$ is the total interference computed by RTA-LC-EDF in the sub problem window with a length x.

- For any $A_k > \overline{A}_k$ (Equation (5)), there is $\Omega \leq m(A_k + D_k - C_k)$ (please check [4] for more details), and this implies $\Omega(x, A_k) \leq m(A_k + D_k - C_k)$.

In the end, if Equation (3) in Bar holds (i.e. $\Omega \leq m(A_k + D_k - C_k)$ and τ_k is schedulable), then there will be $\Omega(x, A_k) \leq m(A_k + D_k - C_k)$ for any $x \in [A_k + C_k, A_k + D_k]$; and there is no way for RTA-LC-EDF to result in a response time upper bound larger than D_k.

The dominance of RTA-LC-EDF over BC is rather straightforward. For any $x \geq A_k + C_k$, it is easy to see that $I_{i,k}(x - A_k) \geq I^{CI}_{i,k}(x - A_k, 0)$, where $I_{i,k}(x)$ is the interference estimation by BC. Thus, $\Omega(x, A_k) \leq \Omega_2 \leq \sum_{i \neq k} I_{i,k}(x - A_k) + mA_k$. In the end, RTA-LC-EDF will return a WCRT upper bound no larger than the one returned by BC. □

4.3 Upper bound to A_k

The RTA-LC-EDF test in Theorem 3 does not provide a bounded range of A_k values for the analysis. In order to apply the new test, we must find a finite set of A_k such that it is enough to estimate the response time upper bound by simply using these A_k values. This is what we are going to present in this section. We remind that A_k denotes the length of the busy period (before the release of target job).

As a first step, we restrict to valid A_k's values for RTA-LC-EDF, where the concept of a valid busy period length is defined below.

Definition 1. ***(Valid** A_k**)*** *Given a busy period, we say it has a valid length A_k if the solution of following iteration would be larger than A_k.*

$$X \leftarrow \left\lfloor \frac{W(X)}{m} \right\rfloor \tag{10}$$

with $W(X) =$

$$\sum_{\tau_i \in \mathcal{T}} W_i^{NC}(X, \mathcal{L}) + \max_{the\ m-1\ largest} W_i^{DIFF}(X, \mathcal{L}). \tag{11}$$

The $W(X)$ formulated above represents the total workload in a sub problem window with length X. A busy period with a valid A_k means that all processors are always occupied in the interval $[s_0, s_0 + A_k)$ by higher-priority jobs and the resulting workload must cause an interference on the execution of target job. Thus, in the analysis, we exclude those busy period lengths such that the solution of Equation (10) is no larger than its corresponding A_k. In the special case that there is no valid A_k, it is safe to conclude that the WCRT of the target task τ_k is C_k.

Then, we explore the two following properties to find an upper bound to A_k.

Theorem 5. *It is sufficient to bound the busy period length A_k by $A_k < A_k^\alpha$ such that*

$$A_k^\alpha = \frac{C_\Sigma + \sum_{\tau_i \in \mathcal{T}} (T_i - C_i)U_i}{m - U_{tot}}.$$

Proof. We first formulate a *generic NC* workload function for a task τ_i in a time interval t without restricting to specific scheduling policy.

$$w_i(t) = \left\lfloor \frac{t}{T_i} \right\rfloor C_i + [\![t \bmod T_i]\!]^{C_i}$$

A linearised upper bound for $w_i(t)$ is the following:

$$lw_i(t) = U_i t + (T_i - C_i)U_i$$

It can be easily seen that $W_i^{NC}(A_k, A_k + D_k) \leq lw_i(A_k)$, and $W_i^{CI}(A_k, A_k + D_k) \leq lw_i(A_k) + C_i$. Thus, in order to have a busy period with valid length A_k, a necessary condition is that

$$C_\Sigma + \sum_{\tau_i \in \mathcal{T}} lw_i(A_k) > m \cdot A_k$$

In the end, we have $A_k < A_k^\alpha$. □

Theorem 6. *It is sufficient to bound the busy period length A_k by $A_k < A_k^\beta$ such that*

$$A_k^\beta = \frac{C_\Sigma + \sum_{\tau_i \in \mathcal{T}} (T_i - D_i)U_i + (U_{tot} - U_k)D_k}{m - U_{tot}}. \tag{12}$$

Proof. This is an extension of the Theorem 3 in [4]. The proof is similar to the proof of A_k^α, in which we use an over-approximation of the DBF function (as in Equation (1)) instead of the linearisation of the workload function. □

As a result, the smaller one between A_k^α and A_k^β is a safe upper bound on busy period length. Moreover, starting from $A_k = 0$, only those A_k values at which $\mathsf{DBF}_i(A_k + D_k)$ changes for some τ_i need to be considered. Finally, the RTA-LC-EDF test has a pseudo-polynomial time complexity: $O(n^3\mathcal{L}_{max}^2\mathcal{N})$ such that $\mathcal{L}_{max}$ is the maximum length of the extended problem window and $\mathcal{N}$ denotes the maximum number of A_k points checked.

In the case of $m = 1$, there is no pessimism in the computation of workload and interference for RTA-LC-EDF. Given that it takes only valid busy periods into account, RTA-LC-EDF is compatible with Spuri's RTA [20] for EDF scheduling in single processor and returns exact worst-case response time when $m = 1$.

Discussion To better understand the differences among the three tests Bar, BC and RTA-LC-EDF, it is useful to see what happens when they are applied to the case of $m = 1$.

As a matter of fact, BC provides only an upper bound to the response time of a task even for $m = 1$, because it analyses just the problem window and not the whole busy period. To the best of our knowledge, RTA-LC-EDF is the first response-time analysis for G-EDF that reduces to the exact Spuri's algorithm [20] for single-processors when $m = 1$.

The Bar schedulability test is based on the demand bound function, so it does not provide a response time. In fact, for the case of $m = 1$, Bar is equivalent to the classical single processor demand-bound analysis [6], which is a necessary and sufficient test. Therefore, we can say that the main difference between Bar and RTA-LC-EDF is that the first provides a yes/no answer for schedulability, whereas the second additionally provides an upper-bound to the response time of a task (which becomes exact for $m = 1$).

On the other hand, we rely on more precise estimation of workload and interference to preserve the dominance over Bar and BC. The fine-grained workload computation of Algorithm 1 and 2 is more precise than the one used in Bar in the extended problem window. The total interference upper bound Ω_2 in Equation (7) is also a key factor and guarantees that RTA-LC-EDF's estimated interference on the target job is never more than the one computed by BC.

Furthermore, in Bar and RTA-LC-EDF, we need to advance the beginning of the problem window and this could result in additional pessimism: *such pessimism is inherent to limited CI techniques* and cannot be completely avoided; however, the computation of Ω_2 reduces it considerably.

4.4 An over-approximate variant of RTA-LC-EDF

Here, we propose an over-approximation of the RTA-LC-EDF test of Theorem 3. We aim to improve the run-time efficiency, while preserving certain guarantees

of the schedulability result.

Given the sub problem window $[s_0, s_1)$ (Figure 3), we further concentrate on its later part after the target job's release, that is $[a, s_1)$. We denote $y = s_1 - a$.

Then, we define $\Omega(y) = \max_{\forall A_k}\{\Omega(A_k + y, A_k) - mA_k\}$. $\Omega(y)$ represents an upper bound on the total interference over interval $[a, s_1)$. In the following, we are interested in knowing if $\Omega(y)$ is large enough such that τ_k's target job cannot complete its worst-case execution in $[a, s_1)$; and we can still upper bound A_k values to the smaller one between A_k^α and A_k^β. In the new test, we skip the validity check of A_k (Equation (10)) for efficiency concern.

Moreover, we employ another bound on A_k that is dependent on the value of y. That is, given any y, we are going to find (if there exists) the first A_k such that the resulting interference does not leave enough space in the interval $[a, s_1)$ for the target job's worst-case execution.

Lemma 1. *In order to decide whether the target job is eligible to finish its worst-case execution within a time interval $[a, s_1)$ with length y, it is enough to consider A_k values $\leq A_k^y$ such that $\left\lfloor \frac{\Omega(A_k^y + y, A_k^y) - mA_k^y}{m} \right\rfloor > y - C_k$.*

Proof. If inequality $\left\lfloor \frac{\Omega(A_k^y + y, A_k^y) - mA_k^y}{m} \right\rfloor > y - C_k$ holds, it means that there exists $A_k = A_k^y$ such that the target job cannot finish execution inside $[a, a + y)$. □

Theorem 7. ***(RTA-LC-EDF-B)*** *An upper bound of τ_k's WCRT is the solution of the following iterative procedure starting from $Y = \phi$ with $\phi = C_k$.*

$$Y \leftarrow \phi + \left\lfloor \frac{\Omega(Y)}{m} \right\rfloor \tag{13}$$

Proof. This can be proved using the same technique as in the proof of Theorem 3. We skip the complete proof for space concerns.

□

The complexity of RTA-LC-EDF-B is $O(n^3 D_{max}^2 \mathcal{N})$, where D_{max} is the maximum deadline among all tasks. However, the value of A_k^y is usually very small. This means that the overall number of A_k values checked in each step of Equation (13) could be very low too, and this would further benefit the run-time performance of RTA-LC-EDF-B.

As for the precision guarantee in RTA-LC-EDF-B, the following theorem proves that it still dominates Bar and BC.

Theorem 8. *RTA-LC-EDF-B $\succeq$ Bar and RTA-LC-EDF-B $\succeq$ BC.*

Proof. For any A_k and $C_k \leq y \leq D_k$, there is $\Omega(A_k + y, A_k) - mA_k \leq \Omega - mA_k$ and $\Omega(y) \leq \sum_{i \neq k} I_{i,k}(y)$, where Ω and $\sum_{i \neq k} I_{i,k}(y)$ are the total interference upper bounds by Bar and BC respectively. Then, following the same reasoning procedure as in the proof of RTA-LC-EDF's dominance over Bar and BC (Theorem 4), it holds that RTA-LC-EDF-B $\succeq$ Bar and RTA-LC-EDF-B $\succeq$ BC. □

In the end, since Bar is optimal in single processors, so is RTA-LC-EDF-B. That is, in case $m = 1$, a task is EDF schedulable if and only if the WCRT estimation returned by RTA-LC-EDF-B is no larger than that task's deadline.

5 Suspension-Aware Schedulability Analysis

Now, we are going to consider tasks that could suspend their executions. We remind that now a task set $\mathcal{T}$ is explicitly separated into self-suspending tasks $\mathcal{T}^s$ and computational tasks $\mathcal{T}^c$ (see Section 2 for more details). For simplicity, we require that the size of $\mathcal{T}^c$ is at least $(m-1)$.

5.1 Suspension-aware schedulability: prior results

The state-of-the-art suspension-aware test for multiprocessor G-EDF scheduling in hard real-time systems is by Liu and Anderson [19], and we denote this test as LA. LA makes an extension from Bar and still relies on its extended problem window (see Figure 3).

A target job from τ_k is chosen for analysis and we use $s_{k,e} \in [0, S_k]$ to denote the cumulative suspension time of this job; such suspension time is needed to bound interference suffered by the target job within the extended problem window. For a computational task, $s_{k,e} = 0$ can be always assumed.

$$I_{i,k}^{NC} = \begin{cases} \left[\!\left[I_i^{NC}(A_k + D_k) \right]\!\right]^{A_k+D_k-C_k-s_{k,e}+1} & \text{if } i \neq k \\ \left[\!\left[I_i^{NC}(A_k + D_k) - C_k \right]\!\right]^{[\![A_k+D_k-T_k]\!]_0} & \text{if } i = k \end{cases}$$

$$I_{i,k}^{CI} = \begin{cases} \left[\!\left[I_i^{CI}(A_k + D_k) \right]\!\right]^{A_k+D_k-C_k-s_{k,e}+1} & \text{if } i \neq k \\ \left[\!\left[I_i^{CI}(A_k + D_k) - C_k \right]\!\right]^{[\![A_k+D_k-T_k]\!]_0} & \text{if } i = k \end{cases}$$

When bounding the total interference, a key difference between task sets with and without self-suspending tasks is that all self-suspending tasks can bring carry-in workload in the beginning of the extended problem window. In the end, together with the fact that there are at most $(m-1)$ computational tasks with CI workload, the total interference in the extended problem window becomes $\Omega =$

$$\sum_{\tau_i \in \mathcal{T}^s} I_{i,k}^{CI} + \sum_{\tau_j \in \mathcal{T}^c} I_{j,k}^{NC} + \sum_{\text{the } m-1 \text{ largest}} I_{j,k}^{DIFF}$$

Theorem 9 (Adapted from Theorem 2 in [19]). *A task set $\mathcal{T} = \mathcal{T}^s \cup \mathcal{T}^c$ is schedulable with G-EDF if, $\forall \tau_k \in \mathcal{T}$, $\forall s_{k,e} \in [0, S_k]$ and for any $0 \leq A_k < \overline{A}_k$, the following holds:*

$$\Omega \leq m(A_k + D_k - C_k - s_{k,e}) \tag{14}$$

with $\overline{A}_k = \frac{m \cdot (C_k + s_{k,e}) + \sum_{\tau_i \in \mathcal{T}} C_i}{m - U_{tot}} - D_k$.

5.2 RTA-LC-EDF(-B) with suspension-awareness

Here, we discuss how to apply RTA-LC-EDF(-B) to systems with self-suspensions. Again, a target job of τ_k with suspension time $s_{k,e}$ is considered.

From suspension-oblivious context to suspension-aware analysis, a series of adaptations are listed below.

- The suspension of a task does not contribute to its worst-case workload. Thus, the workload formulation of a task, which could be from either $\mathcal{T}^s$ or $\mathcal{T}^c$, in Algorithm 1 and Algorithm 2 is still valid.

- In order for the target job (with suspension) to successfully terminate, there should be enough processing time left for both its execution (C_k) and suspension ($s_{k,e}$). That is, when bounding the interference from workload, we should consider the target job's suspension.

$$I_{i,k}^{NC}(x, A_k) = \left[\!\left[W_i^{NC}(x, \mathcal{L}) \right]\!\right]^{x - C_k - s_{k,e} + 1}$$
$$I_{i,k}^{CI}(x, A_k) = \left[\!\left[W_i^{CI}(x, \mathcal{L}) \right]\!\right]^{x - C_k - s_{k,e} + 1}$$

- All self-suspending tasks are regarded as CI tasks. The total interference in the sub problem window is bounded by two estimates: Ω_1 in Equation (6) and Ω_2 in Equation (7), where Ω_1 is obtained by limiting the number of CI tasks in the beginning of sub problem window. As all self-suspending tasks are CI tasks now, we need to refine Ω_1's formulation.

$$\Omega_1 = \sum_{\tau_i \in \mathcal{T}^s} I_{i,k}^{CI}(x, A_k) + \sum_{\tau_j \in \mathcal{T}^c} I_{j,k}^{NC}(x, A_k) + \max_{\text{the } m-1 \text{ largest}} I_{j,k}^{DIFF}(x, A_k)$$

  Similarly, the total workload $W(x)$ within the sub problem window formulated in Equation (11) is refined as follows.

$$W(x) = \sum_{\tau_i \in \mathcal{T}^s} W_i^{CI}(x, \mathcal{L}) + \sum_{\tau_j \in \mathcal{T}^c} W_j^{NC}(x, \mathcal{L}) + \max_{\text{the } m-1 \text{ largest}} W_j^{DIFF}(x, \mathcal{L})$$

- In the suspension-aware context, all self-suspending tasks may bring CI workload into a problem window. Hence, we re-compute A_k^{α} as

$$\frac{C'_{\Sigma} + \sum_{\tau_i \in \mathcal{T}^s} C_i + \sum_{\tau_i \in \mathcal{T}} (T_i - C_i) U_i}{m - U_{tot}}$$

  and A_k^{β} as

$$\frac{C'_{\Sigma} + \sum_{\tau_i \in \mathcal{T}^s} C_i + \sum_{\tau_i \in \mathcal{T}} (T_i - D_i) U_i + (U_{tot} - U_k) D_k}{m - U_{tot}}$$

with C'_Σ denoting sum of the $(m-1)$ largest WCETs among computational tasks.

On the other hand, to decide if the interference $\Omega(y)$ is too large so that the target job's execution, together with its self-suspension, exceeds the interval $[a, s_1)$, it is enough to bound A_k by the following A_k^y with

$$\left\lfloor \frac{\Omega(A_k^y + y, A_k^y) - mA_k^y}{m} \right\rfloor > y - C_k - s_{k,e}.$$

- The last step to adapt RTA-LC-EDF and RTA-LC-EDF-B for **S**uspension-**A**wareness (SA) context is a new starting point, which should take into account the suspension time $s_{k,e}$, for their respective RTA procedures. That is, $\phi = C_k + s_{k,e}$ for Equation (9) and Equation (13).

In the end, the adapted tests are denoted as RTA-LC-EDF-SA and RTA-LC-EDF-SA-B. Instead of enumerating all possible $s_{k,e}$ in $[0, S_k]$ for the WCRT of the target task, we prove that it is enough to only consider the job with maximum suspension time S_k.

Theorem 10. *The WCRT of τ_k can be upper bounded when its target job has $s_{k,e} = S_k$.*

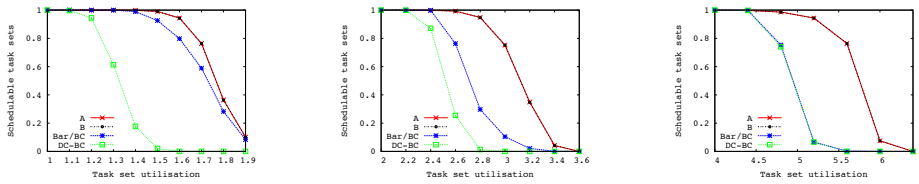
Proof. Let us first have a look at RTA-LC-EDF-SA. Suppose that $\phi_1 = C_k + s_{k,e}$ and $\phi_2 = C_k + s'_{k,e}$ are two starting points (let us say $A_k = 0$) for Equation (9) in the suspension-aware context and $s_{k,e} > s'_{k,e}$. Given two sub problem windows with length X_1 and X_2 corresponding to ϕ_1 and ϕ_2 respectively and $X_1 - X_2 = s_{k,e} - s'_{k,e}$, for any task, the upper bound on its interference in the two cases is $X_1 - \phi_1 + 1 = X_2 - \phi_2 + 1$. That means, if we choose an arbitrary iteration step, the difference between the results starting from ϕ_1 and ϕ_2 is at least $(s_{k,e} - s'_{k,e})$. By assuming $s_{k,e} = S_k$, the resulting WCRT estimation would be a safe upper bound.

For RTA-LC-EDF-SA-B, a similar proof can be conducted. □

In the end, we prove the dominance of the new suspension-aware algorithms over LA.

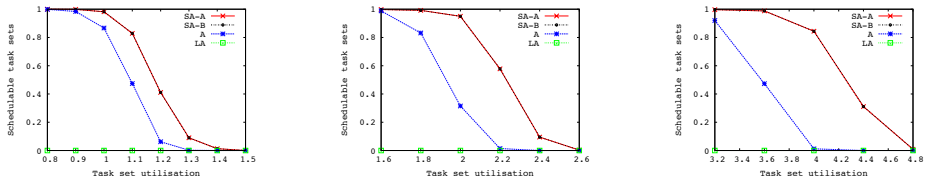
Theorem 11. *RTA-LC-EDF-SA $\succeq$ RTA-LC-EDF-SA-B $\succeq$ LA.*

Proof. Here we only sketch the proof for RTA-LC-EDF-SA-B $\succeq$ LA. Similarly to the suspension-oblivious situation, when explicitly taking the self-suspending time $s_{k,e}$ into account, we have $\Omega(A_k + y, A_k) - mA_k \leq \Omega - mA_k$; $\Omega(A_k + y, A_k)$ and Ω are estimated by RTA-LC-EDF-SA-B and LA (Theorem 9) respectively. Thanks to the more precise estimation of workload and interference, RTA-LC-EDF-SA-B $\succeq$ LA. □



(a) $m = 2$ (b) $m = 4$ (c) $m = 8$

Figure 5: Simulation results for tasks **without** self-suspension



(a) $m = 2$ (b) $m = 4$ (c) $m = 8$

Figure 6: Simulation results for tasks **with** self-suspension

6 Performance Evaluation

In this part we evaluate the performance of different schedulability tests with or without explicitly taking task suspension time into account. For simplicity, we use A and B as abbreviations for RTA-LC-EDF and RTA-LC-EDF-B, respectively; SA-A and SA-B denote the suspension-aware counterparts. Recently, [15] developed a "divide and conquer" technique and applied it to BC, denoted as DC-BC (more specifically, we refer to the NEW-$\mathsf{C}_{\mathsf{EDF}}$ in [15]). In brief, DC-BC judges the target task τ_k to be schedulable if there exists an integer pair configuration $(c', l) \in [0, C_k] \times [0, D_k]$ dividing τ_k into two parts such that both parts satisfy certain interference-based conditions derived from BC. We include DC-BC in the comparison.

Due to space limitation, here we focus on major trends when reporting simulation results[2].

6.1 Tasks without self-suspension

Each task set in the simulation is characterised by a tuple (m, n, U_{tot}) such that m is the number of processors, n is the number of tasks and U_{tot} is the total task utilisation. We consider $m \in \{2, 4, 8\}$, and we set $n = 10 \cdot m$. For

[2]An extended version of this paper, which contains the comprehensive comparision, is available online: https://sites.google.com/site/theyoucheng/rtss15.

each (m, n, U_{tot}) configuration, we randomly generate 1000 task sets. For a task set with some U_{tot}, tasks' utilisations are generated according to Randfixedsum algorithm [12]. Task periods are randomly sampled with uniform distribution in the range $[10, 1000]$. Then, the WCET of a task τ_i is simply computed as $T_i \cdot U_i$ and the relative deadline D_i is randomly chosen with uniform distribution in a range $[0.8 \cdot T_i, T_i]$.

For each generated task set, different schedulability tests are applied, and we record the the number of schedulable task sets returned by each test. Results are reported in Figure 5. The x-axis denotes the total task utilisation U_{tot} and the y-axis represents the percentage of schedulable task sets detected by different tests. The line marked with Bar/BC counts the task sets that are found schedulable by at least one of the two.

Clearly RTA-LC-EDF and RTA-LC-EDF-B substantially improve over all the other tests.

6.2 Tasks with self-suspension

We evaluated the performance of our new suspension-aware tests SA-A and SA-B. Task sets are generated in the same way as before; given any task τ_i, its maximum suspension time S_i is then chosen in the range $[0.5 \cdot C_i, C_i]$. Besides comparing with the state-of-the-art suspension-aware test LA, we also apply RTA-LC-EDF in the suspension-oblivious way, i.e., by simply treating self-suspending time as task execution.

Results are in Figure 6. Notice that LA is very pessimistic and gives positive results for very few task sets. The suspension-aware version of RTA-LC-EDF further improves over the suspension-oblivious version.

6.3 Run-time efficiency

As we have seen from simulation results, the performances of RTA-LC-EDF and of its over-approximation RTA-LC-EDF-B are rather close to each other. However, RTA-LC-EDF-B has a better run-time performance, as shown in Figure 7, where we report the running time of the two algorithms on task sets with $m = 8$, $n = 10 \cdot m$ and different utilisations. Notice that the running time of RTA-LC-EDF-B is up to 100 times shorter than that of RTA-LC-EDF.

7 Conclusion

In this paper, we provided more accurate algorithms for computing the response times of real-time tasks scheduled by G-EDF. We also extended the methodology to self-suspending tasks. Our algorithms strictly dominate the state-of-the-art algorithms, and substantial performance improvements are confirmed by simulation results. In the future, we plan to extend the algorithm to other task models, taking into account interaction through shared resources.

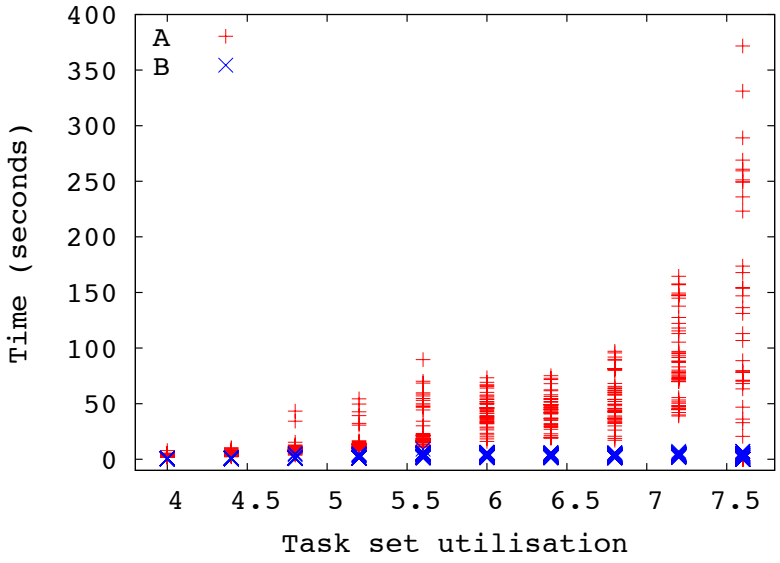


Figure 7: Run-time comparison between the new tests.

References

[1] Theodore Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. *IEEE Real-Time Systems Symposium (RTSS)*, 2003.

[2] Theodore Baker and Sanjoy Baruah. An analysis of global EDF schedulability for arbitrary-deadline sporadic task systems. *Real-Time Systems*, 43(1):3–24, 2009.

[3] Theodore P Baker and Michele Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *Principles of Distributed Systems*, pages 62–75. Springer, 2007.

[4] Sanjoy Baruah. Techniques for multiprocessor global schedulability analysis. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 119–128, 2007.

[5] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. Implementation of a speedup-optimal global EDF schedulability test. In *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*, pages 259–268. IEEE, 2009.

[6] Sanjoy K Baruah, Aloysius K Mok, and Louis E Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190. IEEE, 1990.

[7] Marko Bertogna and Sanjoy Baruah. Tests for global EDF schedulability analysis. *Journal of Systems Architecture*, 57(5):487–497, 2011. Special Issue on Multiprocessor Real-time Scheduling.

[8] Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 149–160. IEEE, 2007.

[9] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pages 209–218. IEEE, 2005.

[10] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 63(4):763–780, 2012.

[11] Robert Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2011.

[12] Paul Emberson, Roger Stafford, and Robert Davis. Techniques for the synthesis of multiprocessor tasksets. In *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, 2010.

[13] Gilles Geeraerts, Joël Goossens, and Markus Lindstrõm. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm. *Real-Time Systems*, 49(2):171–218, 2013.

[14] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 387–397. IEEE, 2009.

[15] Jinkyu Lee. Time-reversibility of schedulability tests. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 294–303. IEEE, 2014.

[16] Jinkyu Lee and Insik Shin. Limited carry-in technique for real-time multicore scheduling. *Journal of Systems Architecture*, 59(7):372–375, 2013.

[17] Jinkyu Lee, Kang G. Shin, Insik Shin, and Arvind Easwaran. Composition of schedulability analyses for real-time multiprocessor systems. *IEEE Trans. Computers*, 64(4):941–954, 2015.

[18] Juri Lelli, Dario Faggioli, Tommaso Cucinotta, and Giuseppe Lipari. An experimental comparison of different real-time schedulers on multicore systems. *Journal of Systems and Software*, 85(10):2405–2416, 2012.

[19] Cong Liu and James H Anderson. Suspension-aware analysis for hard real-time multiprocessor scheduling. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 271–281. IEEE, 2013.

[20] Marco Spuri. Analysis of deadline scheduled real-time systems. 1996.

[21] Youcheng Sun and Giuseppe 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*, page 35. ACM, 2014.

[22] Youcheng Sun, Giuseppe Lipari, Nan Guan, and Wang Yi. Improving the response time analysis of global fixed-priority multiprocessor scheduling. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–9. IEEE, 2014.

[23] Guangmo Tong and Cong Liu. Supporting read/write applications in embedded real-time systems via suspension-aware analysis. In *Embedded Software (EMSOFT), 2014 International Conference on*, pages 1–10. IEEE, 2014.