Improving the Response Time Analysis of Global Fixed-Priority Multiprocessor Scheduling

Youcheng Sun*, Giuseppe Lipari*†, Nan Guan‡§ and Wang Yi§,
Scuola Superiore Sant'Anna*, LSV - ENS Cachan and CNRS†, Northeastern University, China‡, Uppsala University§
{y.sun, lipari}@sssup.it, guannan@ise.neu.edu.cn, yi@it.uu.se

***Abstract*—In this paper we address the problem of schedulability analysis for a set of sporadic tasks with arbitrary deadlines running on a multiprocessor system with global fixed-priority preemptive scheduling.**

We prove the existence of a class of *critical instants* for releasing a task, one of which results in the worst-case response time of that task.

Then, we propose a new analysis that improves over current state-of-the-art Response Time Analysis (RTA) by reducing its pessimism. We also observe that, in the case of unconstrained deadlines, the current RTA may underestimate the carry-in workload, and we propose a new formulation that corrects the problem. Finally, we evaluate the performance improvement of our new response time analysis method by empirical experiments with randomly generated task sets. Experimental results show that our new analysis method can successfully accept a considerable amount of task sets that have to be treated as unschedulable by existing methods.

I. Introduction

The rapid development of multi-core processors leads to an increasing trend of deploying real-time embedded systems on multi-core platforms. This trend demands efficient techniques to schedule real-time workloads on parallel computing architectures. The analysis of *global multiprocessor scheduling* is a significantly more difficult problem than uniprocessor scheduling, as noted by Liu in 1969 [1]: "*The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.*"

The analysis techniques of *fixed-priority* (FP) scheduling on uniprocessor are built upon the concept of *critical instant*, an instant at which a request for the considered task will have the largest response time. It is known that on uniprocessors the critical instant occurs when *all* competing tasks simultaneously release computation requests [2]. Unfortunately, the critical instant of global FP scheduling is generally unknown; the above synchronous release pattern does not necessarily lead to the worst-case response time. Therefore, brute-force enumeration of all the possible run-time release patterns is required for exact schedulability analysis, which is not applicable to realistic-size systems due to serious state-space explosion.

Techniques based on approximation have been proposed to efficiently decide the schedulability of global FP scheduling. A common approximation strategy is to add a safe margin (the so-called *carry-in*) to each competing task's workload in the considered time interval. The state-of-the-art analysis method for this problem is RTA-LC proposed by Guan et al. [3], which limits the number of competing tasks with carry-in to the number of processors minus one. This greatly reduces the pessimism when counting the competing tasks' workload, and yields the best analysis precision among all existing schedulability analysis methods. RTA-LC has been borrowed by related fields such as the analysis of WirelessHART Networks [4].

In this paper we revisit RTA-LC and address both pessimism and optimism problems in its current form:

- We prove that the limited carry-in technique used by RTA-LC indeed corresponds to a certain form of critical instant in global FP scheduling, even for the unconstrained-deadline task systems where the unfinished computation request may accumulate over the next release time.
- We reformulate the calculation of the workload of carry-in tasks. On one hand, our new formulation is more precise than the original calculation in RTA-LC, and more interestingly, leads to the (somehow counter-intuitive) fact that a task with carry-in actually may contribute *fewer* workload than the case without carry-in. On the other hand, our new formulation fixes the potential underestimation of the workload of carry-in tasks in RTA-LC when they bring more than one carry-in jobs into the considered time interval.
- We propose a more precise method to handle the problem of unknown categorisation of carry-in and non-carry-in tasks in the response time bound calculation procedure.

These insights result in a new response time analysis method, called **RTA-CE** (RTA with Carry-in tasks Enumeration).

We evaluate the performance improvement of our new response time analysis method over existing methods by comparing the acceptance ratio with a large amount of randomly generated task sets. It is shown that our new response time analysis method can successfully accept a considerable amount of task sets that have to be treated as unschedulable by existing methods.

A. Related Work

Andersson et al. [5] derived utilisation bounds for fast schedulability test of global FP scheduling. Baker [6] developed sophisticated schedulability analysis techniques by limiting the maximal amount of carry-in workload of each individual task. Bertogna et al. [7] discovered that for each

competing task with very high workload in the considered time interval, the part of its workload that has to execute in parallel with the analysed task should not be taken into account in the actual interference. This yields a more precise schedulability test than all previous tests. Bertogna and Cirinei [8] later applied this technique to iterative response time analysis of global scheduling, which greatly improves analysis precision over previous response time analysis techniques [9], [10], [11] for the same problem.

Guan et al [3] developed RTA-LC (Response Time Analysis with Limited Carry-in), the state-of-the-art technique for schedulability analysis of global FP scheduling. RTA-LC integrates Bertogna and Cirinei's response time analysis and Baruah's technique for global EDF scheduling of limiting the number of carry-in tasks. One important discovery in [3] is that the worst-case response time upper bound of a task can be found in an interval starting at an instant when all processors are occupied by higher priority tasks and before which at least one processor is idle. Later [12] proved that, for *constrained-deadline* tasks, such a release pattern indeed will lead to the exact worst-case response time of a task. In this paper we prove the same fact also holds for *unconstrained-deadline* task systems.

II. System Model

We consider global fixed-priority scheduling on a multiprocessor system. A system consists of a task set $\mathcal{T}$ with n sporadic tasks that run on m processors. Each task is denoted as $\tau_i = (C_i, D_i, T_i)$ where C_i is the worst-case execution time (WCET), D_i is the relative deadline, and T_i is the minimum interarrival time between two successive releases of the task (sometimes also called *period*). A task is said to have a constrained deadline if its relative deadline is smaller than or equal to its period; otherwise, if it has relative deadline larger than period, it is said to be an unconstrained-deadline task. In this paper we consider arbitrary deadlines, i.e. D_i can be smaller than, equal to or larger than the period T_i. And the utilization of a task is defined as $U_i = \frac{C_i}{T_i}$. Every task is assigned a fixed and unique priority. By convention, tasks with lower index have higher priorities: $i < j$ implies that τ_i has higher priority than τ_j.

A real-time task τ_i releases a sequence of *jobs* $J_{i,k}$, with $k = 0, 1, \ldots$. A job $J_{i,k}$ is characterised as follows:

- $r_{i,k}$ is the job's release (or arrival) time;
- $d_{i,k} = r_{i,k} + D_i$ is the job's absolute deadline;
- $f_{i,k}$ is the job's finishing time;
- $R_{i,k} = f_{i,k} - r_{i,k}$ is the job's response time.

We say that a task is active if it has been released but has not completed its execution. The release of a higher priority task can preempt the execution of a lower priority task. In practical implementations of global multiprocessor schedulers, successive jobs of the same task are not allowed to execute in parallel. Therefore, we assume that job $J_{i,k+1}$ cannot start executing until the previous job $J_{i,k}$ has completed. We assume a discrete time model, where all task parameters are non-negative integers and all events (e.g., job release, job finishing) occur at integer time instants.

The worst-case response time (WCRT) R_i of a task is defined as the largest response time among all its jobs. A task τ_i is schedulable if and only if $R_i \leq D_i$.

III. Critical Instants for Global FP Scheduling

In this section, we define the concept of "critical instants" for a task τ_k and we prove that the worst-case response time of τ_k happens when it is released precisely at one of these critical instants. This notion has already been proposed by Guan et al. [3]; later Davis et al. [12] independently proved that, for constrained-deadline tasks, releasing a task at one of its "critical instants" will result in the WCRT.

Here we prove it directly for the general case of arbitrary deadlines.

Clearly, it makes no sense to discuss the critical instants for the m highest priority tasks, since they can execute whenever eligible and their WCRTs simply equal to their WCETs. In the following context, when using the notion of critical instants and response time analysis, we only refer to tasks other than the m highest priority ones.

Definition 1. *A critical instant for task τ_k is an instant t such that:*

- *At least m tasks with priority higher than τ_k are active at t;*
- *At most $m-1$ tasks with priority higher than τ_k are active just before t.*

The critical instant defined above does *not* precisely correspond to a concrete system state at runtime. Instead, it covers a set of possible system states, one of which will lead to the worst-case response time as we will prove in the following. In particular, there can be up to $m-1$ higher priority tasks executing before t and we do not know which are these tasks, when these tasks have been activated and how much of their computation time is left at t.

Apart from the critical instants defined above, we also claim that the worst-case response time of a task occurs when this task releases jobs "as fast as possible", which is formally captured by the following definition:

Definition 2. *Given a sporadic task τ_k, a* chain of consecutive jobs *(or* chain *for short) $J_{k,s}, \ldots, J_{k,h}$, with arrival times $r_{k,s}, \ldots, r_{k,h}$, respectively, is defined as follows:*

- *The job preceding $J_{k,s}$ (if any) has completed before $r_{k,s}$;*
- *every job $J_{k,j}$, $j = s, \ldots, h-1$ completes after $r_{k,j+1}$.*

The chain is said to be tight *if $\forall j = s+1, \ldots h$, $r_{k,j} = r_{k,j-1} + T_k$.*

Note that every job in the system is included in some chain. For instance, in the special case of a constrained-deadline task with ($D_k \leq T_k$), every tight chain only contains one job.

Lemma 1. *The worst-case response time of τ_k is found with a job $J_{k,h}$ in a* tight *chain.*

Proof: We prove the lemma by showing that for an arbitrary chain $J_{k,s}, \ldots, J_{k,h}$ of task τ_k, if we shift the release time of every job $J_{k,j}$ in the chain to $r'_{k,j} = r_{k,s} + (j-s)T_i$, thus obtaining a tight chain, the finishing time of every job $f_{k,j}$, $j = s, \ldots, h$ does not change.

A job cannot start executing until the prior job of the same task has terminated. However, in the original chain, the finishing time of every job (except the last one) is after the arrival time of the successive job. Therefore, when job $J_{k,j}$ arrives at $r_{i,j}$, it must still wait until $f_{k,j-1}$ before it can be activated. By shifting its arrival time to $r'_{k,j} = r_{k,s} + (j-s)T_k$ this precedence relationship is maintained. Therefore, nothing in the schedule changes, and the finishing time of each job in the chain remains the same. ∎

We now prove the critical instant defined in Definition 2 indeed leads to the worst-case response time of a task.

Theorem 1. *The worst-case response time of τ_k is found with a job $J_{k,h}$ in a* tight chain *$J_{k,s}, \ldots, J_{k,h}$, such that $r_{k,s}$ coincides with a critical instant for task τ_k.*

Proof: First of all, by Lemma 1, if the job with the worst-case response time is not in a tight chain, we can find another job, whose response time is not smaller, in a tight chain. So in the following we only focus on jobs in tight chains.

We will show that, if worst-case response time of task τ_k occurs in a tight chain $J_{k,s}, \ldots, J_{k,h}$ that does not start at a critical instant, by modifying the arrival times of the chain we can construct a tight chain $J_{k,s}, \ldots, J_{k,h}$ starting at a critical instant, such that the new response time $R'_{k,h}$ of job $J_{k,h}$ in the resulting chain is no smaller than its counterpart $R_{k,h}$ in the original chain.

The release time of the first job in the original chain does not coincide with a critical instant, so we have two possibilities:

- Suppose that at $r_{k,s}$ there are no more than $m-1$ higher priority tasks active. Let $[A, B]$ be the first interval after $r_{k,s}$ such that there are at least m higher priority active tasks in every instant of $[A, B]$. It is easy to see that $A < f_{k,h}$, because otherwise we could set $r'_{k,s}$ and obtain a chain with a higher response time, as τ_k cannot execute in $[A, B]$.
  Therefore, τ_k executes in $[r_{k,s}, A]$. If we set $r'_{k,s} = A$, the new finishing time of $J_{k,h}$ is
  $$f'_{k,h} \geq f_{k,h} + (A - r_{k,s})$$
  So the new response time of $J_{k,h}$ is
  $$\begin{aligned} R'_{k,h} &= f'_{k,h} - r'_{k,h} \\ &\geq f_{k,h} - (r_{k,s} - A) - r'_{k,h} \\ &= f_{k,h} - (r_{k,s} - A) - (A + (h-s)T_k) \\ &= f_{k,h} - (r_{k,s} + (h-s)T_k) = f_{k,h} - r_{k,h} = R_{k,h} \end{aligned}$$
  Therefore, the new response time is no smaller than the original one.
- Suppose that at $r_{k,s}$ and just before $r_{k,s}$ there are at least m higher priority tasks active. Let A be the latest time before $r_{k,s}$ such that there are no more than $m-1$ higher priority active tasks at $A-1$. By setting $r'_{k,s} = A$, we observe that task τ_k cannot execute in $[A, r_{k,s}]$, because all processors are busy executing higher priority tasks. Also, after $r_{k,s}$ the schedule does not change, and particularly $f'_{k,h} = f_{k,h}$. So the new response time of $J_{k,h}$ is
  $$\begin{aligned} R'_{k,h} &= f'_{k,h} - r'_{k,h} \\ &= f_{k,h} - (r_{k,h} - (r_{k,s} - A)) \\ &= R_{k,h} + (r_{k,s} - A) > R_{k,h} \end{aligned}$$
  i.e., the new response time is strictly larger than the original one.

∎

IV. Revisting RTA-LC

The RTA-LC (Response Time Analysis with Limited Carry-in) in [3] is the most accurate algorithm for response time analysis of global FP scheduling on multiprocessors. In this section we briefly review RTA-LC and identify both the pessimism and optimism in it.

To analyze the schedulability of a task τ_k, we need to take all higher priority tasks $\tau_1, \ldots, \tau_{k-1}$ into account and compute the interference that they may cause on τ_k. To do this, we consider a time interval of length x containing the jobs of τ_k under analysis, namely the *problem window*. By Theorem 1, we can restrict the analysis to a problem window starting at a critical instant for τ_k.

The *workload* of higher-priority task τ_i in a window of length x is the amount of computation that τ_i requires within this time interval. A task τ_i is called a *carry-in task* (CI) if at least one job of τ_i has been released before the beginning of the window and executes within the window. If no such job exists, the task is called *non-carry-in task* (NC). We use $W_i(x)$ to denote an upper bound on the workload of task τ_i over any time interval of length x. More specifically, we will use the notation $W_i^{CI}(x)$ if τ_i is a CI task, and W_i^{NC} if τ_i is a NC task.

The *interference* of a higher-priority task τ_i on τ_k in the problem window is the cumulative length of the intervals during which jobs of τ_i execute and jobs of τ_k have been released but cannot be executed. The upper bound $I_k(\tau_i, x, h)$ is the interference that τ_i causes on the first h jobs of τ_k in the problem window: $I_k^{CI}(\tau_i, x, h)$ for a CI task τ_i and $I_k^{NC}(\tau_i, x, h)$ for a NC task τ_i. When $h = 1$, we often ignore h in the representation; and such an abbreviation also applies to other notations in the paper.

For the task τ_k under analysis, we use $\mathcal{T}$ to denote the set of tasks with higher priority than τ_k. And $\mathcal{T}^{CI} \subset \mathcal{T}$ is the set of carry-in tasks. By Theorem 1, there are at most $m-1$ carry-in tasks in a problem window starting at a critical instant. However, we do not know which subset of tasks causes the worst possible interference on τ_k. We denote with $\mathcal{Z}$ the set of all possible carry-in task sets with no more than $m-1$ higher priority tasks.

Lemma 2. *(Lemma 2 in [3]) The workload bounds can be computed with*

$$W_i^{NC}(x) = \left\lfloor \frac{x}{T_i} \right\rfloor \cdot C_i + [x \bmod T_i]^{C_i} \quad (1)$$

$$W_i^{CI}(x) = \left\lfloor \frac{[x - C_i]_0}{T_i} \right\rfloor \cdot C_i + C_i + \alpha \quad (2)$$

where $\alpha = [[x - C_i]_0 \bmod T_i - (T_i - R_i)]_0^{C_i - 1}$.

Following the notation in [3], $[A]_B = \max(A, B)$, $[A]^C = \min(A, C)$, and $[A]_B^C = [[A]_B]^C$.

Lemma 3. *Let τ_k be a task under analysis, and let τ_i be a higher priority task. If task τ_k is schedulable, then an upper bound to the interference that τ_i causes to the first h instances of τ_k in a problem window of length x ($x \geq h \cdot C_k$) is given by:*

$$I_k^{NC}(\tau_i, x, h) = \min(W_i^{NC}(x), x - h \cdot C_k + 1) \quad (3)$$

$$I_k^{CI}(\tau_i, x, h) = \min(W_i^{CI}(x), x - h \cdot C_k + 1) \quad (4)$$

Given a time interval x, the total interference $\Omega_k(x, h)$ on the first h jobs of task τ_k is defined as :

$$\max_{\mathcal{T}^{CI} \in \mathcal{Z}} \left(\sum_{\tau_i \notin \mathcal{T}^{CI}} I_k^{NC}(\tau_i, x, h) + \sum_{\tau_i \in \mathcal{T}^{CI}} I_k^{CI}(\tau_i, x, h) \right) \quad (5)$$

$\Omega_k(x, h)$ upper bounds the interference from higher priority tasks to the first h jobs of τ_k in the problem window. $\Omega_k(x, h)$ can be computed in linear time, since it is sufficient to find the $m - 1$ maximal values of the difference $I_k^{CI}(\tau_i, x, h) - I_k^{NC}(\tau_i, x, h)$. Given the workload formulation in Lemma 2, I_k^{CI} is never smaller than I_k^{NC}.

Theorem 2. *(Theorem 2 in [3]) For each $h \geq 1$, let $\mathcal{X}^h$ be the minimal solution of the following Equation by doing an iterative fixed point search of the right hand side starting with $x = h \cdot C_k$.*

$$x = \left\lfloor \frac{\Omega_k(x, h)}{m} \right\rfloor + h \cdot C_k \quad (6)$$

then,

$$R_k = \max_{h \in [1, H]} \{\mathcal{X}^h - (h - 1) \cdot T_k\} \quad (7)$$

is an upper bound of τ_k's WCRT.

The H in (7) is an upper bound to the values of h that need to be checked to get the maximal response time bound R_k. The response time analysis procedure terminates as long as the obtained $\mathcal{X}^h - (h - 1) \cdot T_k$ is no larger than T_k. It has been proved in [3] that a bounded H exists to guarantee the termination of the response time analysis procedure if task τ_k satisfies

$$\sum_{i<k} V_i^k + M \times U_k \neq M, \text{ where } V_i^k = \min(U_i, 1 - U_k) \quad (8)$$

A. Pessimism and Optimism in RTA-LC

1) Pessimism in the iteration procedure: Let us first consider the following task set running on two processors: $\tau_1 = (28, 50, 50)$, $\tau_2 = (13, 30, 30)$, $\tau_3 = (5, 50, 50)$, $\tau_4 = (6, 30, 30)$, and $\tau_5 = (6, 40, 40)$. Following Theorem 2, we compute the WCRT upper bound of first four tasks and obtain: $R_1 = 28$, $R_2 = 13$, $R_3 = 18$ and $R_4 = 24$. While iteratively computing the WCRT of task τ_5 and along with x increasing, the CI task set that corresponds to the maximised $\Omega_5(x)$ varies. For example, when $x = 31$, the total interference $\Omega_5(x)$ is maximised with τ_4 as the CI task; and when $x = 38$, $\Omega_5(x)$ is maximised if τ_3 is the CI task. Finally, the iteration in Theorem 2 will report a deadline miss for τ_5. However, this task set is indeed schedulable, according to our improved analysis method that will be introduced in next section.

We now formalize the pessimism in the iteration procedure of RTA-LC. For simplicity we focus on the constrained-deadline case, but the pessimism also exists in the unconstrained-deadline case. Let $\mathcal{T}^{CI_1}$ and $\mathcal{T}^{CI_2}$ be two possible candidates of the carry-in task set. Suppose $\mathcal{T}^{CI_1}$ is indeed the carry-in task set that leads to the worst-case response time of the analysed task τ_k, then during the whole analysis procedure the total interference to task τ_k can be bounded by

$$\Omega_k^1(x) = \sum_{\tau_i \notin \mathcal{T}^{CI_1}} I_k^{NC}(\tau_i, x) + \sum_{\tau_i \in \mathcal{T}^{CI_1}} I_k^{CI}(\tau_i, x)$$

suppose with the above calculation of $\Omega_k^1(x)$ the fixed-point iteration procedure converges at x_1.

Similarly, if $\mathcal{T}^{CI_2}$ is indeed the carry-in task set that leads to the worst-case response time of the analysed task τ_k, then during the whole analysis procedure the total interference to task τ_k can be bounded by

$$\Omega_k^2(x) = \sum_{\tau_i \notin \mathcal{T}^{CI_2}} I_k^{NC}(\tau_i, x) + \sum_{\tau_i \in \mathcal{T}^{CI_2}} I_k^{CI}(\tau_i, x)$$

and the fixed-point iteration procedure converges at x_2.

In general, $\Omega_k^1(x)$ and $\Omega_k^2(x)$ may not dominate each other. In other words, we may have $\Omega_k^1(x') > \Omega_k^2(x')$ for some x' and $\Omega_k^1(x'') < \Omega_k^2(x'')$ for some x''. In particular, we assume it holds

$$\Omega_k^1(x_1) < \Omega_k^2(x_1)$$
$$\Omega_k^2(x_2) < \Omega_k^1(x_2)$$

Since the interference upper bound $\Omega_k(x)$ calculated by (5) upper bounds both Ω_k^1 and Ω_k^2, we know

$$\Omega_k^1(x_1) < \Omega_k(x_1)$$
$$\Omega_k^2(x_2) < \Omega_k(x_2)$$

Therefore, we can conclude that in RTA-LC (using Ω_k calculated by (5)), the fixed-point iteration may not converge at either x_1 or x_2, but converges at some point which is larger than both x_1 and x_2.

2) Pessimism in the carry-in workload of constrained-deadline tasks: Given a task $\tau_i = (19, 50, 50)$ with $R_i = 20$ and $x = 29$. By using Lemma 2, there is $W_i^{CI}(x) = 19$. However, since we know τ_i is running before the problem window, we can observe that the workload of τ_i in a window of length 29 cannot exceed 18. As the minimum time interval between a job's finishment and the successive job's arrival for τ_k is $T_i - R_i = 30$.

3) Optimism in the carry-in workload of unconstrained-deadline tasks: Since we allow $D_i > T_i$, it could be $R_i > T_i$. Suppose a task τ_i is a CI task and $\left\lceil \frac{R_i}{T_i} \right\rceil = 3$. This means at the same moment, there could be at most 3 active jobs from τ_i in the system. And given a problem window with length equivalent to T_i, the workload upper bound of τ_i should be $\geq 3 \cdot C_i - 1$. But the computation result returned by Lemma 2 is $2 \cdot C_i - 1$.

V. RTA-CE: Improved Response Time Analysis

In this section we introduce a new response time analysis that solves the problems described above.

A. New Workload Upper Bound

We start by proposing a new upper bound on the workload.

Lemma 4. *The workload bound of a carry-in task τ_i in the problem window of length x can be computed as:*

$$W_i^{CI}(x) = W_i^{NC}([x - x_p]_0) + [x]^{\delta} \tag{9}$$

where x_p and δ are defined as:

$$x_p = C_i - 1 + \left\lceil \frac{R_i - C_i}{T_i - C_i} \right\rceil T_i - R_i \tag{10}$$

$$\delta = \left\lceil \frac{R_i - C_i}{T_i - C_i} \right\rceil C_i - 1 \tag{11}$$

and W^{NC} is calculated as in (1)*.*

Proof:

We use w to denote the number of carry-in jobs of τ_i. Suppose t is the starting time of the problem window, then by the definition of critical instant, there are at most $m - 1$ tasks having active jobs at time $t - 1$, so the carry-in jobs are already executing at $t - 1$, and the total unfinished execution demand of the carry-in jobs by the start of the problem window is bounded by $w \cdot C_i - 1$.

We argue that the worst-case release pattern leading to the maximal workload of τ_i is as follows: (1) the earliest carry-in job executes as late as possible and finishes at exactly its worst-case response time, (2) the earliest carry-in job starts execution one time unit before the starting point of the problem window, (3) inside the problem window all jobs are released as fast as possible. The argument follows the same strategy as in [6], [3], showing that if we shift the release times in any way, this will not increase the total workload of the job inside the problem window. We omit the detailed reasoning procedure, but we use Figure 1 to depict this worst-case scenario. In the following, we show that with this particular release pattern, the workload of τ_i in the problem window is bounded by Equation (9).

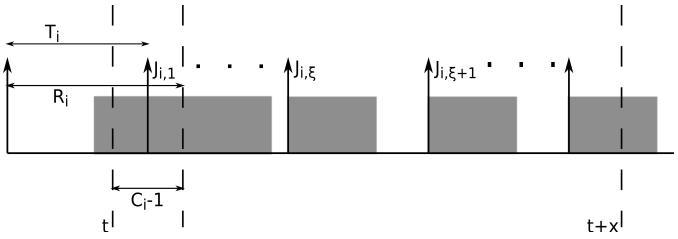


Fig. 1: The CI workload of a task.

We index the job released inside problem window as $J_{i,1}, J_{i,2}, \ldots$, and we assume ξ be the smallest index (if there exists) such that, when $J_{i,\xi}$ is released, all the preceding jobs have been finished. Therefore, the time interval between the starting time of the problem window and the release time of $J_{i,\xi}$ must be at least enough to execute all its preceding jobs. On the other hand, the release time of the ξ^{th} job in the problem window is $x_p = (C_i - 1) + (w + \xi - 1) \cdot T_i - R_i$. So we have

$$w \cdot C_i - 1 + (\xi - 1)C_i \leq (C_i - 1) + (w + \xi - 1) \cdot T_i - R_i$$

Since ξ is an integer, the minimum value for it is:

$$\xi = \left\lceil \frac{R_i - C_i}{T_i - C_i} \right\rceil - (w - 1)$$

Then we can obtain the expression of x_p as in Equation (10).

The total workload of τ_i in the problem window is contributed by two parts: (i) the jobs executing *before* the release of the ξ^{th} job; (i) the jobs executing *after* the release of the ξ^{th} job.

The jobs executing before the ξ^{th} job include the carry-in jobs (the workload of which in the problem window is bounded by $w \cdot C_i - 1$ as we discussed above), and the $\xi - 1$ jobs that are released in the problem window. So their total workload is bounded by

$$\delta = w \cdot C_i - 1 + (\xi - 1)C_i = \left\lceil \frac{R_i - C_i}{T_i - C_i} \right\rceil C_i - 1$$

On the other hand, the maximal workload actually executed in the problem window is bounded by the length of the problem window x.

When τ_i releases a job at x_p, all its preceding jobs have been finished, so the total workload of jobs released in $[x_p, x]$ is bounded by

$$W_i^{NC}([x - x_p]_0)$$

Putting the workload upper bound of the two parts together establishes the lemma. ∎

Our new formulation (9) for calculating $W_i^{CI}(x)$ fixes both the pessimism and optimism issues in Lemma 2. Moreover, our new calculation of $W_i^{CI}(x)$ leads to an interesting fact: now, when $R_i < T_i$, the two functions $W_i^{CI}(x)$ and $W_i^{NC}(x)$ are in general incomparable. More specifically, $W_i^{CI}(x)$ may be smaller than $W_i^{NC}(x)$ with certain x. This can be demonstrated by the following example. Consider the analysis of the workload of task τ_i with $C_i = 2$, $T_i = 4$ and $R_i = 3$,

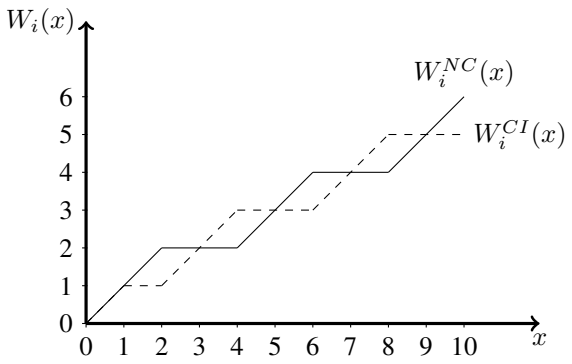


Fig. 2: The worst-case workload of a task with $C_i = 2$, $T_i = 4$ and $R_i = 3$

whose workload function is depicted in Figure 2. We have $W_i^{CI}(2) = 1 < W_i^{NC}(2) = 2$. This (somehow counter-intuitive) phenomenon is due to the fact that a carry-in job has at least been executed for one time unit before the start of the problem window. However, when $R_i \geq T_i$, the relation $W_i^{CI}(x) \geq W_i^{NC}(x)$ still holds.

Finally, we can still compute the carry-in and non-carry-in interference $I_k^{CI}(\tau_i, x, h)$ and $I_k^{NC}(\tau_i, x, h)$ of a higher-priority task τ_i to the analysed task τ_k based on $W_i^{CI}(x)$ and $W_i^{NC}(x)$ by (3) and (4).

B. New Iterative Analysis Procedure

Given a specific carry-in task set $\mathcal{T}^{CI}$, we define the total interference (over time interval x) on task τ_k as follows :

$$\Omega_k(\mathcal{T}^{CI}, x, h) = \sum_{\tau_i \notin \mathcal{T}^{CI}} I_k^{NC}(\tau_i, x, h) + \sum_{\tau_i \in \mathcal{T}^{CI}} I_k^{CI}(\tau_i, x, h) \tag{12}$$

We denote $\mathcal{X}^{h,\mathcal{T}^{CI}}$ the minimal solution of the following iteration :

$$x = \left\lfloor \frac{\Omega_k(\mathcal{T}^{CI}, x, h)}{m} \right\rfloor + h \cdot C_k \tag{13}$$

Lemma 5. *Given a task τ_k and a carry-in task set $\mathcal{T}^{CI}$, the WCRT of τ_k with respect to $\mathcal{T}^{CI}$ is upper bounded by :*

$$R_k^{\mathcal{T}^{CI}} = \max_{h \in [1,H]} \{\mathcal{X}^{h,\mathcal{T}^{CI}} - (h-1) \cdot T_k\} \tag{14}$$

Proof: We show here the case when deadlines are less than or equal to periods and the arbitrary deadline case can be derived similarly. For simplicity, we use $I_k(\tau_i, x)$, $\Omega_k(\mathcal{T}^{CI}, x)$ and $\mathcal{X}^{\mathcal{T}^{CI}}$ by ignoring $h = 1$.

Suppose $\mathcal{X}^{\mathcal{T}^{CI}}$ is the minimal solution of the iteration

$$x = \left\lfloor \frac{\Omega_k(\mathcal{T}^{CI}, x)}{m} \right\rfloor + C_k \tag{15}$$

We are going to prove $\mathcal{X}^{\mathcal{T}^{CI}}$ is an upper bound of τ_k's response time with respect to $\mathcal{T}^{CI}$.

The proof is similar as in [8]. By contradiction. Suppose, for some carry-in task set $\mathcal{T}^{CI}$, the iteration in Equation (15) ends with a value $\mathcal{X}^{\mathcal{T}^{CI}} \leq D_k$, but the response time of τ_k, released with the same $\mathcal{T}^{CI}$, is higher than $\mathcal{X}^{\mathcal{T}^{CI}}$. Since the iteration ends, there is

$$\mathcal{X}^{\mathcal{T}^{CI}} = \left\lfloor \frac{\Omega_k(\mathcal{T}^{CI}, \mathcal{X}^{\mathcal{T}^{CI}})}{m} \right\rfloor + C_k$$

That is,

$$\mathcal{X}^{\mathcal{T}^{CI}} = \left\lfloor \frac{\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}})}{m} \right\rfloor + C_k \tag{16}$$

Remind that $\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}})$ is a short form for

$$\sum_{\tau_i \notin \mathcal{T}^{CI}} I_k^{NC}(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}}) + \sum_{\tau_i \in \mathcal{T}^{CI}} I_k^{CI}(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}})$$

According to Theorem 3 in [8], if R_k^{ub} is a response time upper bound of τ_k, we have

$$\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, R_k^{ub}) < m(R_k^{ub} - C_k + 1)$$

We assumed that $\mathcal{X}^{\mathcal{T}^{CI}}$ is not an upper bound, hence we reverse the above formula:

$$\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}}) \geq m(\mathcal{X}^{\mathcal{T}^{CI}} - C_k + 1) \tag{17}$$

Then, by replacing $\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}})$ in (16) with the right hand side of (17), we have

$$\begin{aligned} \mathcal{X}^{\mathcal{T}^{CI}} &\geq (\mathcal{X}^{\mathcal{T}^{CI}} - C_k + 1) + C_k \\ &= \mathcal{X}^{\mathcal{T}^{CI}} + 1 \end{aligned}$$

By contradiction, we know that $\mathcal{X}^{\mathcal{T}^{CI}}$ is the upper bound of response time of τ_k with carry-in task set $\mathcal{T}^{CI}$. ■

Theorem 3. *(RTA-CE) The upper bound of WCRT of task τ_k is*

$$R_k = \max_{\mathcal{T}^{CI} \in \mathcal{Z}} \left(R_k^{\mathcal{T}^{CI}} \right) \tag{18}$$

Proof: As long as τ_k is released at one of its critical instants, its response time is then upper bounded by R_k, which, according to Theorem 1, is the upper bound of τ_k's worst-case response time. ■

Note that with our new formulation of computing $W_i^{CI}(x)$ the overall analysis procedure still guarantees to terminate under the condition (8). This follows the similar reasoning in [3].

The WCRT upper bound returned by RTA-CE will never be larger than the result of RTA-LC. As the cost of a more precise response time estimation, the new RTA needs to explicitly enumerate all possible carry-in task sets in $\mathcal{Z}$ (this is where the name RTA-CE comes from) and this leads to an exponential time complexity for the new RTA. However, as shown later in the experiment, the RTA-CE can handle task systems with realistic scales in reasonable time.

C. Improving the Efficiency

The iterative fixed-point calculation in RTA-CE with each given h should start with an initial value no larger than the minimal solution of (13) (otherwise the iteration converges at a larger solution and results in more pessimistic response time bounds). Under this constraint, the initial value should be as large as possible, to make the iteration procedure converge faster. A straightforward and safe initial value is $x = h \cdot C_k$. In the following, we introduce a larger safe initial value to speedup the analysis procedure. We first define:

$$\omega_k(x,h) = \sum_{\tau_i \in \mathcal{T}} \min\{I_k^{NC}(\tau_i, x, h), I_k^{CI}(\tau_i, x, h)\} \quad (19)$$

Note that, as we discussed in Section V-A, W_i^{NC} and W_i^{CI} are generally incomparable, and so do I_k^{NC} and I_k^{CI}. Let s^h be the minimal solution of the following iteration starting with $x = h \cdot C_k$:

$$x = \left\lfloor \frac{\omega_k(x,h)}{m} \right\rfloor + h \cdot C_k \quad (20)$$

For any $\mathcal{T}^{CI}$, it holds $\omega_k(x,h) \leq \Omega_k(\mathcal{T}^{CI}, x, h)$, so we know s^h is no larger than $\mathcal{X}^{h,\mathcal{T}^{CI}}$ for any $\mathcal{T}^{CI}$. Therefore, s^h can be used as a safe initial value for x in Equation (13) for any $\mathcal{T}^{CI}$.

Furthermore, to find $\mathcal{X}^{h,\mathcal{T}^{CI}}$, we could utilise the already computed $\mathcal{X}^{h-1,\mathcal{T}^{CI}}$ and start with $x = C_k + \mathcal{X}^{h-1,\mathcal{T}^{CI}}$.

In conclusion, the starting point for Equation (13) is

$$x = \begin{cases} s^h & h = 1 \\ \max\{s^h, C_k + \mathcal{X}^{h-1,\mathcal{T}^{CI}}\} & h > 1 \end{cases} \quad (21)$$

VI. Experiments

Each test in the experiment is described by a tuple (m, n, U) where m is the number of processors, n is the number of tasks and U is the total task utilisation in the task set. Task sets are randomly generated according to Randfixedsum algorithm in [13]. Every task has its period T_i uniformly distributed in the range $[100, 200]$. For constrained-deadline tasks, the ratio between D_i and T_i is distributed in the range $[0.7, 1]$; for arbitrary-deadline tasks, $\frac{D_i}{T_i} \in [0.7, 1.3]$. Before applying RTA-CE and RTA-LC, tasks in a task set are first assigned priorities by the Deadline Monotonic (DM) strategy, i.e. a task with shorter deadline gets a higher priority.

A. Performance Tests

We conduct experiments with randomly generated task sets to evaluate the precision improvement of RTA-CE over RTA-LC. We consider $m \in \{2, 4\}$, and given m we set its corresponding task set size n and task set utilisation U to be $n = 10 \cdot m$ and $U \in \{0.025m, 0.5m, \ldots, 0.975m, m\}$, respectively. For each configuration (m, n, U), 1000 task sets are randomly generated.

We report the results in Figure 3. The horizontal axis specifies task set utilisations and the vertical axis denotes the number of schedulable systems found. We omit plotting points where the number of schedulable task sets is simply 1000 or 0 in figures.

The experimental results confirm RTA-CE's improvement over RTA-LC in practice. For example, in the case of tasks with arbitrary deadlines where $m = 2$, $n = 20$ and $U = 1.35$, RTA-CE finds 111 more schedulable task sets (among 1000 randomly generated task sets) than RTA-LC.

The advantage of RTA-CE over RTA-LC may not be always as shown in Figure 3. For example, if we assume task periods are distributed in range $[10, 1000]$ and $\frac{D_i}{T_i} \in [0.7, 2]$, we obtain the experimental results in Figure 4. On the other hand, it is possible to set up different experiments and get even stronger dominance relation between RTA-CE and RTA-LC than Figure 3 shows.

B. Efficiency Tests

The efficiency tests contain two parts:

- We investigate how much the efficiency improvement scheme introduced in Section V-C can effect the run-time performance of RTA-CE;
- we demonstrate to which extent RTA-CE can (or cannot) scale through a showcase study.

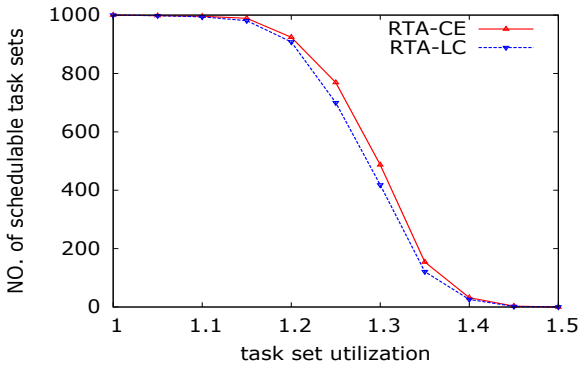
All tests are conducted in a MacBook with 2.5 GHz Intel Core i5 processor.

At first, we discuss the experiment for testing RTA-CE's efficiency strategy. To set up tests, we consider $m = 6$, $n = 60$, $U \in \{3, 4, 5, 6\}$ and arbitrary-deadline tasks. For every (m, n, U), 30 task sets are randomly generated. In the experiment, we measure the time spent for deciding the schedulability of tasks by RTA-CE with efficiency enhancement (RTA-CE-WE) and without efficiency enhancement (RTA-CE-WOE).

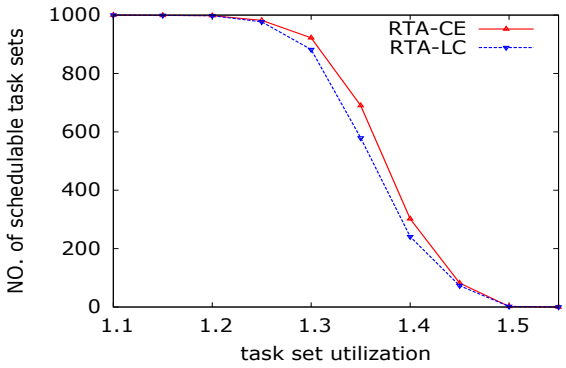
There are 60 different priority levels (1-60) in each generated task set and a lower number means higher priority. RTA procedure is applied from highest priority task (with priority 1) to lowest priority task (with priority 60). For all task sets we report the accumulated time, which starts from applying RTA-CE on the task with priority 1, that is used by RTA-CE-WE and RTA-CE-WOE respectively to decide the schedulability at each priority level. In this way, for a schedulable task set, the time obtained for priority level 60 is actually the time used to decide the schedulability of a task set. For a task set that is not schedulable, we report time instances recorded till the last task that RTA-CE checks.

The final results are plotted in Figure 5. On average, RTA-CE-WE saves $1/5 \sim 1/4$ RTA-CE-WOE's run-time. For instance, the time RTA-CE-WE uses to deal with 60 tasks and the time RTA-CE-WOE spends on 58 tasks are almost the same. And RTA-CE-WE can always obtain the analysis result (schedulable or unschedulable) for a task set in less than 20 minutes.

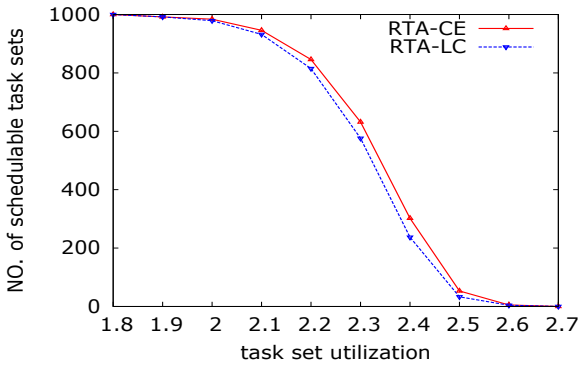
To further stress the efficiency improvements, we randomly generate an arbitrary-deadline task set with $m = 8$, $n = 80$ and $U = 4$. Then we measured the run-time for applying RTA-CE on this task set. The test result is reported in Figure 6. We manually stopped the test after the run-time exceeded 20 hours: at that instant, RTA-CE had finished the schedulability check of the first 63 tasks.



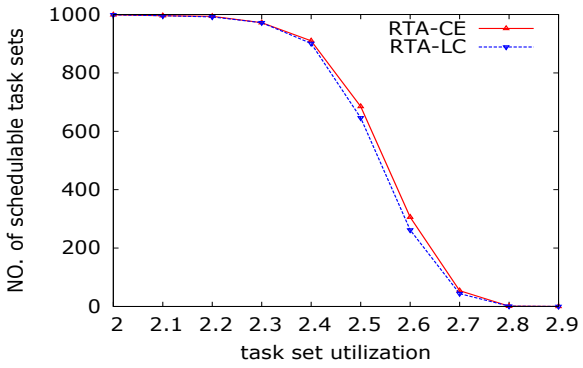
(a) $m = 2, n = 20, \frac{D_i}{T_i} \in [0.7, 1]$



(b) $m = 2, n = 20, \frac{D_i}{T_i} \in [0.7, 1.3]$

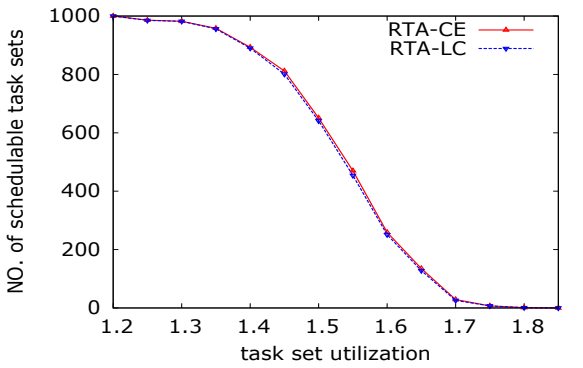


(c) $m = 4, n = 40, \frac{D_i}{T_i} \in [0.7, 1]$

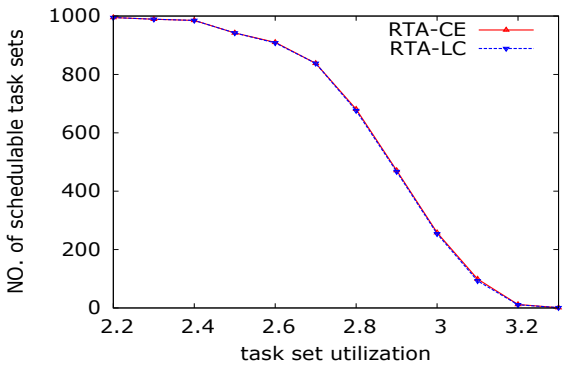


(d) $m = 4, n = 40, \frac{D_i}{T_i} \in [0.7, 1.3]$

Fig. 3: RTA-CE v.s. RTA-LC



(a) $m = 2, n = 20$



(b) $m = 4, n = 40$

Fig. 4: RTA-CE v.s. RTA-LC ($T_i \in [10, 1000]$, $\frac{D_i}{T_i} \in [0.7, 2]$)

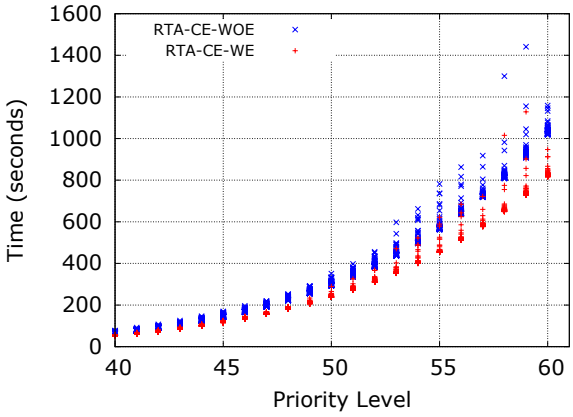


Fig. 5: Efficiency improvement tests

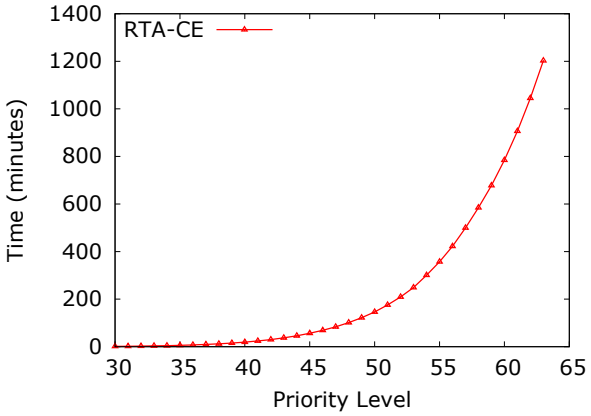


Fig. 6: The scalability test ($m = 8, n = 80, U = 4$)

VII. Conclusion

In this paper we considered the response time analysis problem for global fixed-priority scheduling on multiprocessors. We proved the existence of a type of critical instant leading to the worst-case response time of a task. The idea of this critical instant has been used in the context of approximated analysis, but has not been strictly proved to lead to the actual worst-case response time for arbitrary-deadline task sets.

Then we improved the state-of-the-art technique RTA-LC by addressing both its pessimism and optimism. We first propose a new formula to bound the workload of carry-in jobs. The new formula is, on one hand more precise than the one used in RTA-LC, and on the other hand it fixes the potential underestimation of the carry-in workload for unconstrained-deadline tasks. We then proposed a new iterative response time analysis procedure that achieves better precision. Experiments with randomly generated tasks show that our new method RTA-CE can successfully accept a considerable number of task sets that are deemed to be unschedulable by RTA-LC.

References

[1] C. L. Liu, "Scheduling algorithms for multiprocessors in a hard real-time environment," *In JPL Space Programs Summary*, 1969.

[2] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, January 1973.

[3] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*. IEEE, 2009, pp. 387–397.

[4] A. Saifullah, Y. Xu, C. Lu, and C. Y., "End-to-end delay analysis for fixed priority scheduling in wirelesshart networks," *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[5] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," *IEEE Real-Time Systems Symposium (RTSS)*, 2001.

[6] T. Baker, "Multiprocessor edf and deadline monotonic schedulability analysis," *IEEE Real-Time Systems Symposium (RTSS)*, 2003.

[7] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of edf on multiprocessor platforms," *Euromicro Conference on Real-Time Systems (ECRTS)*, 2005.

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

[9] B. Andersson and J. Jonsson, "Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling," *Technical Report, Chalmers University of Technology*, 2001.

[10] A. Burns and A. Wellings, *Real-time systems and programming languages*. Addison-Wesley, 3rd edition, 2001.

[11] L. Lundberg, "Multiprocessor scheduling of age constraint processes," *RTCSA*, 1998.

[12] R. I. 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.

[13] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, 2010, pp. 6–11.