New Response Time Bounds for Fixed Priority Multiprocessor Scheduling

Nan Guan^1, Martin Stigge^1, Wang Yi^1 and Ge Yu^2

^{1} Uppsala University, Sweden
^{2} Northeastern University, China

Abstract

Recently, there have been several promising techniques developed for schedulability analysis and response time analysis for multiprocessor systems based on over-approximation. This paper contains two contributions. First, to improve the analysis precision, we apply Baruah's window analysis framework [6] to response time analysis for sporadic tasks on multiprocessor systems where the deadlines of tasks are within their periods. The crucial observation is that for global fixed priority scheduling, a response time bound of each task can be efficiently estimated by fixed-point computation without enumerating all the busy window sizes as in [6] for schedulability analysis. The technique is proven to dominate theoretically state-of-the-art techniques for response time analysis for multiprocessor systems. Our experiments also show that the technique results in significant performance improvement compared with several existing techniques for multiprocessor schedulability analysis. As the second main contribution of this paper, we extend the proposed technique to task systems with arbitrary deadlines, allowing tasks to have deadlines beyond the end of their periods. This is a non-trivial extension even for single-processor systems. To our best knowledge, this is the first work for multiprocessor systems in this setting, which involves sophisticated techniques for the characterization and computation of response time bounds.

1 Introduction

There is an increasing interest in developing real-time applications on multiprocessor platforms due to the broad introduction of multicore chip processors. It is also one of the most challenging problems today for the real-time research community to develop efficient techniques for the analysis of such systems.

Recently, there have been several promising techniques developed for schedulability analysis e.g. [6] and response time analysis e.g. [8] for global multiprocessor scheduling. In this paper, we take a second look at the problem of Response Time Analysis (RTA) for multiprocessor systems with global fixed-priority scheduling. We will present a new RTA technique to further improve the analysis precision of the existing techniques and also a non-trivial extension of the technique to task systems with arbitrary deadlines for multiprocessor systems.

Roughly speaking, RTA is to estimate the response time bound for each task in a set of tasks when they are scheduled using a given scheduling policy. It is an important technique for the design and analysis of not only hard real-time systems as it may be used for schedulability analysis but also soft real-time systems as the response time bounds provide an indication on how a system performs. For single processor systems, RTA has been intensively studied in the past two decades, and extended to various task models [11, 17, 2, 15, 20], to deal with real-life systems. Today we have obtained a rather good understanding of RTA for single processor scheduling. In contrast, much less work on RTA for multiprocessor scheduling has been done by now.

Our work is mainly built upon the work of [8] and [6]. We apply the window analysis framework of [6] to response time analysis inspired by the work of [8] for sporadic tasks on multiprocessor systems with both constrained and arbitrary-deadline tasks. The crucial observation is that when the earliest time instant, after which all processors are occupied with tasks of higher priorities, occurs *just before* the release of a task, it results in an upper bound of the worst-case response time of the task for global fixed priority scheduling. This allows us to efficiently compute the bound by fixed-point computation without enumerating all the busy window sizes as in [6] for schedulability analysis. Comparing with previous work, we contribute in the following two ways:

- We have significantly improved the analysis precision compared to the state-of-the-art techniques. It is not only shown theoretically but also demonstrated in experiments.
- To our best knowledge, this is the first work on RTA for global multiprocessor scheduling with arbitrary-deadline task systems.

The paper is structured as follows: Section 2 introduces the task model and used notations. Section 3 reviews the previous work on RTA and outlines our contributions. Section 4 presents our proposed RTA technique for task systems with constrained deadlines, and then Section 5 extends the RTA technique to the arbitrary deadline case. Section 6 presents the performance evaluation, and conclusions are summarized in Section 7. To improve the readability, detailed proofs are given in the appendix.

2 Problem Setting

We consider global fixed priority scheduling on a multiprocessor platform consisting of M processors.

A sporadic task set τ consists of N sporadic tasks running on this platform. We use $\tau_i = \langle C_i, D_i, T_i \rangle$ to denote such a task where C_i is the *worst-case execution time* (WCET), D_i is the relative deadline for each release, and T_i is the minimum inter-arrival separation time also referred to as the *period* of the task. We further assume that all tasks are ordered by priorities, i.e., τ_i has higher priority than τ_j iff $i < j$. The *utilization* of a task τ_i is $U_i = C_i/T_i$.

A *constrained-deadline* task τ_i satisfies the restriction $D_i \leq T_i$, whereas an *arbitrary-deadline* task τ_i does not constrain the relation between D_i and T_i. We will consider both types.

A sporadic task τ_i generates a potentially infinite sequence of *jobs* with successive job-arrivals separated by at least T_i time units. We use J_i^h to denote the h-th job of τ_i. We omit the superscript h and just use J_i to denote a job of τ_i if there is no need to identify which job it is. Each job J_i^h adheres to the conditions C_i and D_i of its task τ_i and has additional properties concerning *absolute time points* related to its execution, which we denote with lower case letters. The *release time* is denoted by r_i^h, the *deadline* by d_i^h which is derived by $d_i^h = r_i^h + D_i$, and the *finish time* by f_i^h, which is the time instant at which J_i^h just finished its execution. We define the *response time* of J_i^h as the difference between its release and finish times:

$$R_i^h = f_i^h - r_i^h$$

The worst-case response time (WCRT) R_i of task τ_i is the maximal response time value among all jobs of τ_i in all job sequences possible in the system.

Since D_i is allowed to be larger than T_i, it is possible that several jobs of a task are *active* (i.e., released but not yet finished) simultaneously. We restrict that a job J_i^h can execute only if its precedent job J_i^{h-1} has been already finished, to avoid unnecessary working space conflict. This restriction is commonly adopted in the implementation of real-time operating systems for multicores/multiprocessors, for instance, RTEMS [19] and LITMUS^{RT} [13]. Thus, we define the *ready time* γ_i^h of J_i^h as $\gamma_i^h = \max(r_i^h, f_i^{h-1})$, which is the earliest time instant for a released job J_i^h to execute if no higher-priority task is interfering with it. At all time points $t \in [\gamma_i^h, f_i^h)$ we call J_i^h *ready*. Note that there is at most one ready job of each task at each time point (also for arbitrary deadlines).

We use the discrete time concept, i.e., any time value involved in the scheduling is a non-negative integer. This is based on the assumption that all events in the system happen only at clock ticks. Thus, we use time point t to represent the entire time interval $[t, t+1)$.

Without any loss of generality, we assume that tasks are strictly periodic (i.e., that $r_i^h = r_i^{h-1} + T_i$), unless stated otherwise. However, all the results are also applicable to sporadic task sets in general.

For simplicity of expression, we further use the following notations to express that a value A is "limited" if it is below or above a certain threshold B or C, respectively: $[\![A]\!]_B = \max(A, B)$, $[\![A]\!]^C = \min(A, C)$, and $[\![A]\!]_B^C = [\![[\![A]\!]_B]\!]^C$. This expression just keeps the value A if it is within the interval $[B, C]$, otherwise it is B if $A < B$ or C if $A > C$.

3 Previous Work and Our Contributions

Before presenting our proposed techniques, we briefly review the previous work on RTA, to provide a primary knowledge background to readers that are not familiar with this field, as well as outline the contributions of this paper against previous work.

3.1 The Basic Single-processor Case

The RTA technique was for the first time proposed in [16], where it is only applicable to constrained-deadline task sets (i.e., $\forall \tau_i \in \tau : D_i \leq T_i$).

RTA of a task τ_k is based on the concept of *level-k busy period*. Intuitively, the *level-k busy period* is the maximum continuous time interval during which a processor executes tasks of priority greater than or equal to the priority of the considered task τ_k, until τ_k finishes its active job. For single-processor fixed priority scheduling, the situation exhibiting the worst-case response time, is known to happen at a well-defined *critical instant*: All higher priority tasks are released together with the analyzed task τ_k, i.e., at the same time instant. Thus, the maximal interference suffered by τ_k in a level-k busy period of length x can be computed by $\sum_{i<k} \lceil x/T_i \rceil \cdot C_i$. Using this, τ_k's WCRT can be calculated by finding the minimal solution of the following recursive equation:

$$x = \sum_{i<k} \left\lceil \frac{x}{T_i} \right\rceil \cdot C_i + C_k$$

This solution can be found by interpreting the RHS as a monotonic function in x, whose minimal fixed point can be computed iteratively, starting at $x = C_k$.

3.2 The Basic Multiprocessor Case

RTA has been applied to multiprocessor scheduling with constrained-deadline task systems. The difference to the single-processor case is, that the critical instant in multiprocessor scheduling is generally unknown. This prevents calculation of the exact interference suffered by the analyzed task τ_k during a level-k busy period. Instead, one has to derive an upper bound of the interference.

The work done by a task τ_i in the worst case during a level-k busy period can be divided into three parts:

body: the contribution of all jobs (called *body jobs*) with both release time and deadline *in* the level-k busy period;

carry-in: the contribution of at most one job (called *carry-in job*) with release time *earlier* than the level-k busy period and deadline *in* the level-k busy period;

carry-out: the contribution of at most one job (called *carry-out job*) with release time *in* the level-k busy period and deadline *after* the level-k busy period.

A naive upper bound of the workload of each task τ_i during a level-k busy period of length x is obtained by assuming that the carry-in and carry-out of τ_i both contribute C_i each:

$$W_k^{naive}(\tau_i, x) = \left\lceil \frac{x}{T_i} \right\rceil C_i + C_i$$

Adding the workload of all higher-priority tasks, one can use the term $\frac{1}{M} \sum_{i<k} W_k^{naive}(\tau_i, x)$ as an upper bound of the interference time suffered by τ_k during the level-k busy period of length

x, due to all higher-priority tasks workload. Therefore, an upper bound of the response time of τ_k can be obtained by finding the minimal solution of the following recursive equation [1, 12, 18]:

$$x = \frac{1}{M} \sum_{i<k} \left(\left\lceil \frac{x}{T_i} \right\rceil C_i + C_i \right) + C_k$$

Bertogna and Cirinei [8] have significantly improved the above result. We refer to their RTA technique as [BC-RTA] for short in this paper and give now a short overview of their key ideas. Firstly, rather than assuming that the carry-in and carry-out of a task are both C_i, they derived a more precise upper bound $W_k(\tau_i, x)$ of the workload for each task τ_i which is more precise than $W_k^{naive}(\tau_i, x)$, by carefully identifying the worst-case workload of each individual task.

Secondly, they observed that if the workload of a task τ_i is "too large", not necessarily all its workload can cause interference to the analyzed task τ_k, since the "extra" part of τ_i's workload has to be executed in parallel with τ_k. This is a fundamental difference between single-processor scheduling and multiprocessor scheduling (since in single-processor scheduling, no parallel execution takes place). In particular, they define the *interference* of τ_i to τ_k during a level-k busy period of length x as:

$$I_k(\tau_i, x) = [\![W_k(\tau_i, x)]\!]_0^{x-C_k+1} \quad (1)$$

Using this observation, the recursive equation becomes:

$$x = \left\lfloor \frac{1}{M} \sum_{i<k} I_k(\tau_i, x) \right\rfloor + C_k \quad (2)$$

Note that in Equation (1), the upper bound of τ_i's interference is $(x - C_k + 1)$ rather than $(x - C_k)$. With $(x - C_k)$ as the upper bound, the solution we get from Equation (2) would not be guaranteed to be the upper bound of τ_k's response time. Intuitively, the "+1" is necessary to increase the right hand side of (2) as long as there is more interference that could potentially prevent τ_k from running. For example, when the iterative search for the least fixed point is started with $x = C_k$, the search would stop immediately, since $\min(W_k(\tau_i, x), x - C_k)$ would be 0 for all i. A formal explanation of this issue can be found in [8].

3.3 With Arbitrary Deadlines on Single-processors

The RTA for single-processor scheduling has also been extended to arbitrary-deadline task sets (i.e., $D_i > T_i$ is allowed) [17]. The level-k busy period concept is extended in the following way. It starts at the release time of a job whose previous jobs have been finished at its release time. Without loss of generality, we call the job in question J_k^1 with release time r_k^1. Note that $\gamma_k^1 = r_k^1$, so the job is immediately ready when it is released. The level-k busy period lasts now until a job of τ_k (which we denote with J_k^H) is finished before the next released time, as shown in Figure 1. Note that, it can reach over several of τ_k's periods.

To calculate the response time of each job J_k^h with $h \in \{1, \dots, H\}$, the following recursive equation is solved starting with $h = 1$. Every time a solution is obtained, h is increased by one:

$$x = \sum_{i<k} \left\lceil \frac{x}{T_i} \right\rceil \cdot C_i + h \cdot C_k \quad (3)$$

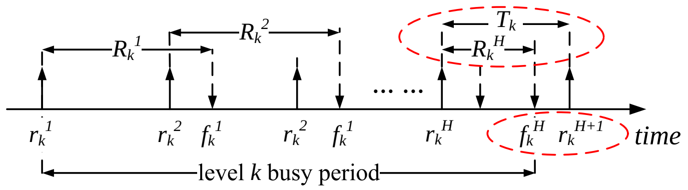


Figure 1. level-k busy period for single-processor scheduling with arbitrary deadlines

Let χ^h denote the solution in step h. The response time of the job J_k^h is

$$R_k^h = \chi^h - (h-1) \cdot T_i,$$

since we need to subtract the periods in which the $(h-1)$ preceding jobs were released. This procedure of repeatedly solving Equation (3) with increasing h continues until the first solution satisfying the termination condition

$$\mathsf{Term}(h, k): \quad \chi^h \leq h \cdot T_k \quad (4)$$

is found. So we have $H = \min\{h \geq 1 \mid \mathsf{Term}(h, k)\}$.

The WCRT of τ_k is the maximal response time of all jobs of τ_k in the level-k busy period. Thus, τ_k's WCRT may be obtained with:

$$R_k = \max_{h \in \{1, \dots, H\}} \{R_k^h\}$$

The question arises whether an index H satisfying the termination condition $\mathsf{Term}(H, k)$ from (4) exists, i.e, if the procedure terminates. Fortunately, it is easy to prove that this is the case if

$$\sum_{i \leq k} U_i \leq 1.$$

To our best knowledge, there is no known work on the RTA *for multiprocessor scheduling* with arbitrary-deadline task sets.

3.4 Our Contributions

Although [BC-RTA], the state-of-the-art RTA technique for multiprocessor scheduling as sketched above in Section 3.2, has exhibited better performance than other schedualbility tests [9, 3], it still has limitations in the following two aspects: (1) The analysis is still too pessimistic. (2) It can not handle task sets with arbitrary deadlines. The contributions of this paper concern both aspects:

1. In Section 4.3, we employ the idea of busy period (problem window) extension from [6] to further reduce the overestimation of the interference without decreasing the analysis efficiency. The proposed RTA technique theoretically dominates [BC-RTA] (any task set accepted by [BC-RTA] can be accepted by ours), and exhibits a significant performance improvement over [BC-RTA] (and all other state-of-the-art schedulability analysis techniques for this problem).

2. We extend our proposed RTA techniques to arbitrary-deadline task sets in Section 5.3. Note that this is a nontrivial extension even in the light of the work for single-processors reviewed in Section 3.3. The termination problem of the analysis proved to be a notable challenge, since it is much more difficult than in the single-processor case.

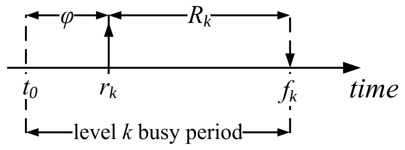


Figure 2. Extended level-k busy period for constrained-deadline task sets

4 Constrained-deadline Task Sets

4.1 Busy Period Extension

In [BC-RTA], to derive a safe upper bound of the interference suffered by the analyzed task τ_k, it is assumed that every higher priority task τ_i has carry-in. This is an over-pessimistic assumption, since in a real scheduling sequence, it may be the case that some task τ_i's carry-in job has finished before the beginning of the busy period, therefore it actually does not contribute any carry-in to the interference of τ_k. To address a similar problem in schedulability tests of global EDF scheduling, Baruah [6] extends the busy period to an earlier time instant, which allows to bound the number of tasks doing carry-in by $M - 1$ (with M being the number of processors), which is in general much smaller than the N (the number of tasks) in [BC-RTA].

In the following we will apply Baruah's idea of busy period extension to RTA. It should be noted, that a trivial combination of the busy period extension and RTA will lead to very high computational complexity, and could fail to yield analysis results for large-scale systems in reasonable time. However, as will be shown later, we can combine them without decreasing the analysis efficiency.

From now on, we let J_k be a job of τ_k that has the worse-case response time. As in [6], we extend the beginning of the level-k busy period from r_k (the release time of J_k), to an earlier time instant t_0, which is defined as the earliest time instant before r_k, such that at any time instant $t \in [t_0, r_k)$ all processors are occupied by tasks with higher priority than τ_k. If there is no such a time instant, we set $t_0 = r_k$. Using this, the *level-k busy period* is defined as the time interval $[t_0, f_k)$, and we define $\varphi = r_k - t_0$, which is the time span by which the busy period extends to the left, as shown in Figure 2.

This definition of t_0 is chosen to impose a bound on the number of tasks contributing carry-in, since at time point $t_0 - 1$, there have to be strictly less than M higher priority tasks active. We state that property as the following lemma:

Lemma 1. *There are at most $M - 1$ tasks having carry-in, and for each task τ_i, the carry-in is at most $C_i - 1$.*

Proof. By discussion above, similar to [6]. □

Comparing with the original level-k busy period, with which one has to assume that all higher-priority tasks have carry-in, the over-estimation of the interference has been significantly reduced.

However, as introduced here, this busy period extension technique is not for free, since it is generally unknown when t_0 actually is, i.e., φ is an unknown variable. To solve this, [6] derives an upper bound (denoted by Φ^B) of all φ's values that need to be checked. The schedulability test is then conducted by enumerating every value of φ in $[0, \Phi^B]$. Finally, J_k (and therefore τ_k) is determined to be schedulable if the test can succeed with every value in $[0, \Phi^B]$.

The upper bound Φ^B is pseudo-polynomial in the values of the task parameters, and in practise usually very large – especially for task sets with large parameter scales and/or with high utilizations. The RTA procedure itself (finding the fixed point) is also generally of pseudo-polynomial complexity, and requires quite a number of iterations in practise. Therefore, if one trivially conducts the RTA on each value of φ in $[0, \Phi^B]$, the complexity of the analysis would be very high in practise, and thus not practically usable. However, as we will see, the RTA procedure in our setting actually needs to be conducted only one single time with $\varphi = 0$, to get the same safe WCRT upper bound as enumerating all possible values of φ.

4.2 Workload and Interference

Before introducing the RTA procedure in detail, we will show bounds for workload and interference of tasks τ_i in a busy period of length x. These will later be used in the analysis.

Workload

The *workload* of a task in a certain busy period is the length of the accumulated execution time of that task within the busy period. We use $W_k(\tau_i, x)$ to denote an upper bound of the workload of each task τ_i with higher priority than the analyzed task τ_k in the level-k busy period of length x. From Lemma 1, we already know that there is at most $M - 1$ tasks doing carry-in, and all other tasks do not provide carry-in. So we define two types of workload:

- $W_k^{\mathsf{NC}}(\tau_i, x)$ denotes the workload bound if τ_i does *not* have a carry-in job;
- $W_k^{\mathsf{CI}}(\tau_i, x)$ is the workload bound if τ_i has a carry-in job.

To compute them, we have the following lemma.

Lemma 2. *The workload bounds can be computed with*

$$W_k^{\mathsf{NC}}(\tau_i, x) = \left\lfloor \frac{x}{T_i} \right\rfloor \cdot C_i + [\![x \mod T_i]\!]^{C_i} \tag{5}$$

$$W_k^{\mathsf{CI}}(\tau_i, x) = \left\lfloor \frac{[\![x - C_i]\!]_0}{T_i} \right\rfloor \cdot C_i + C_i + \alpha \tag{6}$$

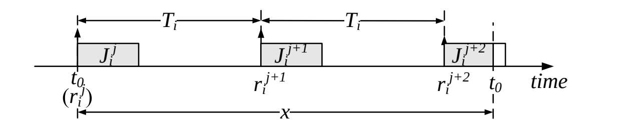
where $\alpha = [\![[\![x - C_i]\!]_0 \bmod T_i - (T_i - R_i)]\!]_0^{C_i - 1}$.

Proof. Similar to reasoning in [6, 14], see Figure 3. □

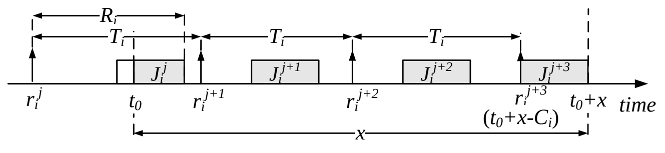
Due to space limitations, we chose not to present a detailed proof and just show some intuition for the computation in Figure 3. We would like to point out two important issues:

First, α in the computation of $W_k^{\mathsf{CI}}(\tau_i, x)$ represents the carry-in of τ_i, which is limited to $C_i - 1$ according to Lemma 1. Further, the carry-in job is guaranteed to finish its computation within its response time R_i. Since we do the RTA for each task in their priority order, a bound of R_i is already known for each higher-priority task τ_i when computing $W_k^{\mathsf{CI}}(\tau_i, x)$ for τ_k.

Second, and most important, we see from Equations (5) and (6) that both $W_k^{\mathsf{NC}}(\tau_i, x)$ and $W_k^{\mathsf{CI}}(\tau_i, x)$ are *independent*



(a) $W_k^{\mathsf{NC}}(\tau_i, x)$



(b) $W_k^{\mathsf{CI}}(\tau_i, x)$

Figure 3. Computing $W_k^{\mathsf{NC}}(\tau_i, x)$ and $W_k^{\mathsf{CI}}(\tau_i, x)$

of φ. This means that, given only the length x of the level-k busy period, we always get the same result of $W_k^{\mathsf{NC}}(\tau_i, x)$ and $W_k^{\mathsf{CI}}(\tau_i, x)$, no matter when t_0 is (i.e., how large φ is). This key observation enables us to greatly reduce the computational efforts necessary do derive the safe WCRT bound.

Interference

As in [BC-RTA], we define the *interference* $I_k(\tau_i, x)$ of τ_i in the level-k busy period of length x. The interference denotes the part of the workload that can actually interfere with τ_k, i.e., can prevent it from running. It can be less than the workload as we already discussed in Section 3.2. By carefully setting bounds, the analysis precision can be greatly improved.

Similar to the workload, we also use two types of $I_k(\tau_i, x)$: We use $I_k^{\mathsf{NC}}(\tau_i, x)$ to denote the bound on the interference of τ_i to τ_k during a busy period of length x if τ_i does *not* have a carry-in job, while we use $I_k^{\mathsf{CI}}(\tau_i, x)$ if τ_i has a carry-in job. Both values can be calculated with:

$$I_k^{\mathsf{NC}}(\tau_i, x) = \llbracket W_k^{\mathsf{NC}}(\tau_i, x) \rrbracket_0^{x-C_k+1} \tag{7}$$

$$I_k^{\mathsf{CI}}(\tau_i, x) = \llbracket W_k^{\mathsf{CI}}(\tau_i, x) \rrbracket_0^{x-C_k+1} \tag{8}$$

As discussed in Section 3.2, the upper bound of the interference needs to be $x - C_k + 1$ rather than $x - C_k$.

We now define the *total interference* $\Omega_k(x)$, as the maximal value of the sum of all higher-priority tasks' interference among all possible cases with

$$\Omega_k(x) = \max_{(\tau^{\mathsf{NC}}, \tau^{\mathsf{CI}}) \in \mathcal{Z}} \left(\sum_{\tau_i \in \tau^{\mathsf{NC}}} I_k^{\mathsf{NC}}(\tau_i, x) + \sum_{\tau_i \in \tau^{\mathsf{CI}}} I_k^{\mathsf{CI}}(\tau_i, x) \right), \tag{9}$$

where $\mathcal{Z} \subseteq \tau \times \tau$ is the set of all partitions of the set $\tau_{<k} = \{\tau_1, \ldots, \tau_{k-1}\}$ into τ^{NC} and τ^{CI}, such that $\tau^{\mathsf{NC}} \cup \tau^{\mathsf{CI}} = \tau_{<k}$, $\tau^{\mathsf{NC}} \cap \tau^{\mathsf{CI}} = \emptyset$ and $|\tau^{\mathsf{CI}}| \leq M - 1$. By taking the maximum over this set, $\Omega_k(x)$ describes the maximal total interference when at most $M - 1$ are having carry-in, and all the others do not have carry-in. According to Lemma 1, $M - 1$ is the maximal number of tasks with carry-in, so indeed, $\Omega_k(x)$ expresses the maximal interference of higher-priority tasks to a task τ_k during a level-k busy period of length x.

Note that $\Omega_k(x)$ can be computed in linear time, since it is sufficient to find the $M - 1$ maximal values of the difference $I_k^{\mathsf{CI}}(\tau_i, x) - I_k^{\mathsf{NC}}(\tau_i, x)$, as pointed out in [6].

We state an important lemma about $\Omega_k(x)$.

Lemma 3. *For all jobs J_k and all $x < f_k - t_0$, the following holds:*

$$\left\lfloor \frac{\Omega_k(x)}{M} \right\rfloor > x - C_k \tag{10}$$

Proof. The proof is given in Appendix A.1. □

Intuitively, the lemma states that, if we let the level-k busy period end before J_k's finish time, the total interference of all higher-priority tasks is large enough to prevent J_k from being finished within the level-k busy period. This in turn indicates that the level-k busy period actually has not reached its end, and thereby should continue to increase. Thus, this property of $\Omega_k(x)$ enables the iterative RTA procedure as will be presented in the next section.

4.3 The RTA procedure

After defining an upper bound $\Omega_k(x)$ of the total interference to a task τ_k in a level-k busy period of length x, we present now how to use this for conducting the response time analysis for τ_k. The level-k busy period begins at the time point t_0, which is φ time units before r_k, the release of τ_k's job (see Section 4.1). In general, φ is an open variable. For a moment, we assume that the length φ of the busy period extension is given, and consider the particular $t_0 = r_k - \varphi$ derived from that. The following lemma expresses the response time analysis for such a particular t_0.

Lemma 4. *Given a $\varphi \geq 0$, let χ be the minimal solution of the recursive equation*

$$x = \left\lfloor \frac{\Omega_k(x)}{M} \right\rfloor + C_k. \tag{11}$$

Then $\chi - \varphi$ is an upper bound of τ_k's response time with this particular $t_0 = r_k - \varphi$.

Proof. Suppose the real worst-case response time of τ_k with $t_0 = r_k - \varphi$ is R, and assume $\chi - \varphi < R$ for the sake of contradiction. Since R is the real WCRT, there is a job sequence in which a job J_k exhibits this response time, i.e., $f_k - r_k = R$. It follows:

$$\chi < R + \varphi = f_k - r_k + \varphi = f_k - t_0$$

Thus, Lemma 3 applies, and therefore (10) holds with $x = \chi$. This contradicts the assumption of χ being a solution of (11). □

Note that for all $k \leq M$, the minimal solution of (11) is trivially C_k, since in that case, $\Omega_k(x) < M$ for $x \leq C_k$ by definition. This matches the intuition that the M highest priority tasks don't suffer any interference, since M processors are available to accommodate them independently.

We have now seen how an upper bound of the response time of τ_k can be derived, if a particular φ is given. Since φ is an open variable, we need to find an upper bound of all response times for all φ to get a safe bound for the response time in general. Naively, it would seem that we have to enumerate all possible values of φ and solve (11), to obtain a safe upper bound of τ_k's WCRT.

However, as mentioned in Section 4.2, the computation of $W_k^{\mathsf{NC}}(\tau_i, x)$ and $W_k^{\mathsf{CI}}(\tau_i, x)$ is *independent of* φ. Thus, it turns out that $\Omega_k(x)$ is also independent of φ. Therefore, no matter what value of φ we are dealing with, the solution of Equation (11) is always the same. And since $\chi - \varphi$ is the upper bound of τ_k's response time (with that particular φ), the maximal value and therefore general response time bound is χ. Thus, we only need to do the RTA according to Lemma 4 with $t_0 = r_k$ (i.e., $\varphi = 0$). The derived solution is guaranteed to be the upper bound of τ_k's WCRT.

Note that this observation can be regarded as a kind of *critical instant* for multiprocessor fixed-priority scheduling. In the context of our analysis, $\varphi = 0$ is guaranteed to be worst among all cases. Namely, we get the worst case when the earliest time instant after which all processors are occupied by higher-priority tasks occurs just before the release of τ_k. Still, since this does not provide precise information about the worst-case release times of the higher priority tasks, we are left with a *set* among which the real critical instant is found – but this set is significantly smaller than the whole space of possible job sequences.

We summarize the conclusion as the following theorem.

Theorem 1 ([OUR-RTA]). *Let χ be the minimal solution of the following Equation (12) by doing an iterative fixed point search of the right hand side starting with $x = C_k$.*

$$x = \left\lfloor \frac{\Omega_k(x)}{M} \right\rfloor + C_k \tag{12}$$

Then χ is an upper bound of τ_k's WCRT.

Proof. By Lemma 4 and the above discussion. □

Note that the iterative fixed point search should terminate with an "unschedulable" result as soon as $x > T_k$, since $D_k \leq T_k$. This also ensures termination of the procedure, even without concrete values of D_k being given.

Since $\Omega_k(x)$ in [OUR-RTA] is no larger than $\sum_{i<k} I_k(\tau_i, x)$ in [BC-RTA], [OUR-RTA] dominates [BC-RTA] in the sense that the upper bound of the WCRT derived by [OUR-RTA] is guaranteed to be no larger than the one derived by [BC-RTA].

5 Arbitrary-deadline Task Sets

We will now extend our RTA technique to the arbitrary-deadlines setting, i.e., the constraint "$D_i \leq T_i$" is dropped now.

5.1 Busy Period

As in the constrained-deadline case from the previous section, we will use a *busy period* whose beginning is extended to an earlier time point t_0 (as in Section 4.1). In addition, we need to extend also its end, since, like in Section 3.3, several jobs could be delayed beyond the next period.

In particular:

- Let J_k^1 be a job of τ_k satisfying $\gamma_k^1 = r_k^1$, i.e., all previous jobs (if any) have been finished at the release of J_k^1.

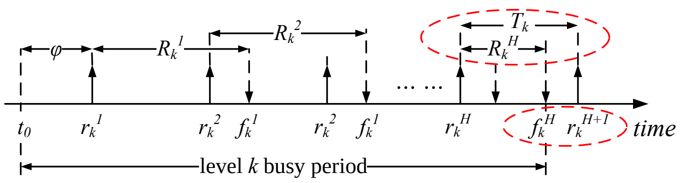


Figure 4. Extended level-k busy period for arbitrary-deadline task sets.

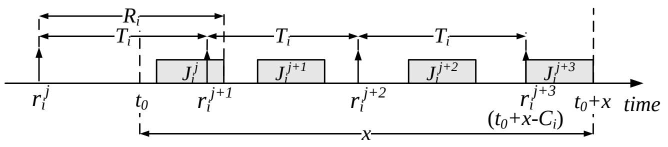


Figure 5. Computing $W_k^{\mathsf{CI}}(\tau_i, x)$ with $R_i > T_i$

- Let t_0 be the earliest time instant before r_k^1, such that at any time instant $t \in [t_0, r_k^1)$ all processors are occupied by tasks with higher priority than τ_k, as before. (Again, if there is no such a time instant, we set $t_0 = r_k^1$.) We also let $\varphi = r_k^1 - t_0$.
- Finally, let J_k^H (with $H \geq 1$) be the first job of τ_k after J_k^1, whose response time is at most τ_k's period, i.e., $R_k^H \leq T_k$. See Figure 4.

For now, we just assume that such a job J_k^H exists. In Section 5.4 we will discuss under which conditions it can be proven to exist.

Using the above, the *level-k busy period* is defined as the time interval $[t_0, f_k^H)$, which spans over the busy period extension φ, plus $H-1$ periods of length T_k each, plus the response time R_k^H of job J_k^H.

Note that Lemma 1 still holds here, i.e., there are at most $M-1$ tasks having carry-in, and the carry-in of each task τ_i is at most $C_i - 1$.

5.2 Workload and Interference

The *workload* calculation of $W_k^{\mathsf{NC}}(\tau_i, x)$ and $W_k^{\mathsf{CI}}(\tau_i, x)$ is the same as the constrained-deadline case, shown in Lemma 2. The only difference we notice here is that in the computation of $W_k^{\mathsf{CI}}(\tau_i, x)$, it can be the case that a task's response time R_i is larger than T_i, as shown in Figure 5. However, the computation of $W_k^{\mathsf{CI}}(\tau_i, x)$ considering this case is still the same as Equation (6).

For the *interference* calculation, the only additional concern is that the extended level-k busy period may contain *several* jobs of τ_k. Thus, let h be the number of jobs of τ_k in a level-k busy period of length x, then we extend the interference bounds for the arbitrary-deadline case (c.f. Equation (7) and (8) in Section 4.2) as follows:

$$I_k^{\mathsf{NC}}(\tau_i, x, h) = \llbracket W_k^{\mathsf{NC}}(\tau_i, x) \rrbracket_0^{x-h\cdot C_k+1} \tag{13}$$

$$I_k^{\mathsf{CI}}(\tau_i, x, h) = \llbracket W_k^{\mathsf{CI}}(\tau_i, x) \rrbracket_0^{x-h\cdot C_k+1} \tag{14}$$

Using these, a bound for the *total interference* $\Omega_k(x, h)$ to h instances of τ_k during a level-k busy period of length x is calculated just as before (c.f. Equation (9)):

$$\Omega_k(x,h) = \max_{(\tau^{\mathsf{NC}},\tau^{\mathsf{CI}})\in\mathcal{Z}} \left(\sum_{\tau_i\in\tau^{\mathsf{NC}}} I_k^{\mathsf{NC}}(\tau_i,x,h) + \sum_{\tau_i\in\tau^{\mathsf{CI}}} I_k^{\mathsf{CI}}(\tau_i,x,h) \right) \tag{15}$$

5.3 The RTA procedure

In this section, we generalize the response time analysis procedure of Section 4.3 to the arbitrary-deadline setting. We will again first assume a given φ, so we consider a particular $t_0 = r_k^1 - \varphi$. The following lemma is the generalization of Lemma 4.

Lemma 5. *Given a $t_0 = r_k^1 - \varphi$, let for each $h \geq 1$ denote χ^h the minimal solution of the recursive equation*

$$x = \left\lfloor \frac{\Omega_k(x,h)}{M} \right\rfloor + h \cdot C_k. \tag{16}$$

Let further $H(\varphi)$ be the minimal index satisfying

$$\chi^{H(\varphi)} \leq H(\varphi) \cdot T_k + \varphi. \tag{17}$$

Then

$$R_k^\varphi = \max_{h\in[1,H(\varphi)]} \{\chi^h - (h-1)\cdot T_k - \varphi\} \tag{18}$$

is an upper bound of τ_k's response time with this particular t_0.

Proof. Similar to Lemma 4, using a generalized version of Lemma 3. □

As in the constrained-deadlines case of Section 4.3, the above was first dealing with the case of a given φ. Again, this is an open variable in general, so naively, the maximal R_k^φ over all φ needs to be found to derive a safe upper bound for the response time in general. It turns out that, as before, it will be only necessary to consider the case $\varphi = 0$ to get a safe bound, without checking other values for the open variable φ.

For the arbitrary-deadlines case in this section, this conclusion is a little bit more involved than the constrained-deadlines case in Section 4.3. The reason is that we are dealing with a *set* of solutions for the recursive equation (16), which could in general look different for different values of φ. Thus, for $\varphi_1 > \varphi_2$, we don't have a straight forward derivation for a relation between $R_k^{\varphi_1}$ and $R_k^{\varphi_2}$ as in the constrained-deadlines case (Section 4.3).

However, we still have the property that $\Omega_k(x,h)$ is independent of φ. Thus, for each h, the minimal solution found for Equation (16) will be independent of φ. Therefore, it follows from (17), that $\varphi_1 > \varphi_2$ implies $H(\varphi_1) \leq H(\varphi_2)$, since the condition is "looser" for larger φ. But this means that for $\varphi_1 > \varphi_2$, the maximum in (18) for calculating $R_k^{\varphi_1}$ is taken over a set whose elements all have a larger counterpart in the set for calculating $R_k^{\varphi_2}$. Thus, we have the following implication:

$$\varphi_1 > \varphi_2 \quad \Rightarrow \quad R_k^{\varphi_1} \leq R_k^{\varphi_2}$$

It follows directly that, in order to maximize R_k^φ, only $\varphi = 0$ needs to be considered, since it is the minimal possible value.

We conclude with a formulation of our response time analysis procedure for arbitrary-deadline task sets in the following theorem, that directly follows from the above discussion. We will use the following two predicates as termination conditions:

- Termination because of job finishing before next period:

$$\mathsf{Term}(h,k) \equiv \left[\chi^h \leq h \cdot T_k\right]$$

- Termination because of detected possible deadline miss:

$$\mathsf{Miss}(h,k) \equiv \left[\chi^h > (h-1)\cdot T_k + D_k\right]$$

Theorem 2 ([OUR-RTA-arb]). *For each $h \geq 1$, let χ^h be the minimal solution of the following Equation (19) by doing an iterative fixed point search of the right hand side starting with $x = h \cdot C_k$.*

$$x = \left\lfloor \frac{\Omega_k(x,h)}{M} \right\rfloor + h \cdot C_k \tag{19}$$

Let H be the minimal index satisfying $\mathsf{Term}(H,k)$, then

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

is an upper bound of τ_k's WCRT.

Proof. By Lemma 5 and the above discussion. □

The procedure can be implemented by conducting the fixed point search for (19) starting with $h = 1$, and repeating with increased h until the termination condition $\mathsf{Term}(h,k)$ holds. Note that during the fixed point search, the procedure should terminate with a "unschedulable" result, as soon as a deadline miss is detected, i.e. $x > (h-1)\cdot T_k + D_k$ (which is similar to $\mathsf{Miss}(h,k)$). This guarantees termination for the fixed point searches.

5.4 Termination

The response time calculation is guaranteed to calculate a safe bound for the WCRT *if* it terminates with $\mathsf{Term}(h,k)$. We will show now, that it can be guaranteed to terminate with that condition in certain cases. In other cases, this can not be guaranteed, but then at least the condition $\mathsf{Miss}(h,k)$ holds eventually, as we will see. (Note that we already showed how to guarantee termination for the fixed point searches themselves. What we are concerned about now is the termination of the whole procedure with repeated invokation of fixed point searches.)

To discuss this termination problem, we define a new metric, the *Interfering Utilization* of τ_i with respect to task τ_k:

$$V_i^k = \min(U_i, 1 - U_k). \tag{20}$$

It can be seen as a generalization of the original utilization metric U_i to the multiprocessor case. Intuitively, it restricts the utilization which is relevant for interference to the part that is not running in parallel to the task in question. (Recall the discussion in Section 3.2 about deriving I_k by limiting W_k.)

With this new metric, we can formulate and prove the following theorem concerning the termination of [OUR-RTA-arb].

Theorem 3. *For each task τ_k satisfying*

$$\sum_{i<k} V_i^k + M \cdot U_k \neq M \tag{21}$$

the [OUR-RTA-arb] *procedure always terminates, i.e., the following holds:*

$$\exists h \geq 1 : \mathsf{Term}(h,k) \vee \mathsf{Miss}(h,k)$$

Proof. The proof is given in Appendix A.2. □

The theorem states that, provided Condition (21), if $\mathsf{Term}(h,k)$ does not hold for any h, then there will be a deadline violation detected (at least overapproximated). Note that even though a deadline specification for τ_k needs to be supplied in order to be able to do that, the termination is not dependent on the actual value of D_k. So in particular, if $\mathsf{Term}(h,k)$ does not hold for any h, then there is no bound of the response time (at least by our over-approximation).

There is one case remaining:

$$\sum_{i<k} V_i^k + M \cdot U_k = M$$

In this case, there are task sets for which [OUR-RTA-arb] terminates and others for which it does not, so termination can not be guaraneed. Since this is a rather special corner case that can be checked before starting the procedure (and possibly the parameters slightly adjusted to fit one of the other two cases), this is not considered a practical problem.

6 Performance Evaluation

We evaluate the performance of the proposed RTA technique in terms of acceptance ratio. We follow the method in [4] to generate task sets: A task set of $M+1$ tasks is generated and tested. Then we iteratively increase the number of tasks by 1 to generate a new task set, and all the schedulability tests are run on the new task set. This process is iterated until the total processor utilization exceeds M. The whole procedure is then repeated, starting with a new task set of $M+1$ tasks, until a reasonable sample space has been generated and tested. This method of generating random task sets produces a fairly uniform distribution of total utilizations, except at the extreme end of low utilization.

The default setting of the experiments whose results we show in Figure 6 is as follows: the priorities are assigned according to global Deadline Monotonic scheduling; the number of processors is 6; for each task τ_i, T_i is uniformly distributed in $[10, 30]$. For each subfigure, the range of U_i and D_i/T_i is tuned (see the caption of each subfigure), to get task sets with different characteristics.

In all the figures, the curve "Sim" denotes the acceptance ratio of simulations. Since it is not computationally feasible to try all possible task release offsets and inter-release separations exhaustively in simulations, all task release offsets are set to be zero and all tasks are released periodically, and simulation is run for the hyper-period of all task periods. Simulation results obtained under this assumption may sometimes determine a task set to be schedulable even though it is not, but they can serve as a coarse upper bound of the ratio of all schedulable task sets.

Figure 6-(a)(b)(c) shows the comparison between [OUR-RTA] and previous work for constrained-deadline task sets. In [8], [BC-RTA] has been compared with all state-of-the-art analysis for constrained-deadline task sets at that time, and shown clear performance improvement. Thus, for the constrained-deadline case, we compare our analysis [OUR-RTA] (denoted by "Our") with [BC-RTA] (denoted by "BC") and the schedulability test from [6][1] (denoted by "Bar"), which is not included in the comparison in [8]. Another recent work by Bertogna et al. [10] is not included in Figure 6 because it is outperformed by [BC-RTA]. It can be seen from the evaluation that [OUR-RTA] has non-trivial performance improvement over the others, especially with task sets with low utilizations.

[1][6] works on global EDF scheduling, however, it can be easily adapted to global fixed priority scheduling.

Figure 6-(d)(e)(f) shows the comparison between [OUR-RTA-arb] and previous work for arbitrary-deadline task sets. In the figures, "Our" denotes [OUR-RTA-arb], "Bak" denotes the schedulabilty test in [3], "Load" denotes the two schedulability tests based on the LOAD function in [5] and [7] (a task is accepted if it can be accepted by one of these two tests). It can be seen that the performance improvement of [OUR-RTA-arb] over the previous work is significant. Especially in (d) with task sets with low utilizations, the acceptance ratio of [OUR-RTA-arb] is quite close to the simulation. Recall that the simulation curve in the figure is just a coarse upper bound of the acceptance ratio of the exact schedulability test, so it is fair to say that [OUR-RTA-arb] is actually very precise for task sets consisting of low-utilization tasks.

We also evaluated the scalability of [OUR-RTA-arb]. We did experiments with parameters as follows: the number of processors is 100; the number of tasks in each task set is uniformly distributed in $[100, 500]$; the period of each task is uniformly distributed in $[100, 1000]$; the utilization of each task is uniformly distributed in $[0.1, 0.3]$ and the ratio between D_i and T_i is uniformly distributed in $[0.8, 4]$. We conducted [OUR-RTA-arb] on 1000 such task sets on a server computer with AMD Opteron 844 (1.8GHz) CPU, and the experiment was finished in about 30 minutes, which indicates that [OUR-RTA-arb] can handle a real-life-scale task system on average in a few seconds.

7 Conclusions

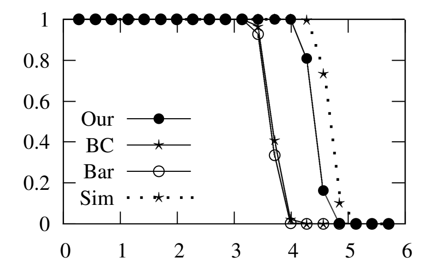
We have developed a new technique to derive response time bounds for global fixed priority scheduling on multiprocessors. This work made contributions in two folds: (1) The analysis precision has been significantly improved against previous work and, (2) To our best knowledge, this is the first work to study the RTA problem of arbitrary-deadline systems on multiprocessors. We have used intensive experiments with randomly generated task sets to evaluate the performance of the proposed analysis techniques, in terms of both precision and efficiency. Experiments show that the proposed analysis has significant improvement of the analysis precision over existing methods, and can easily handle real-life-scale task systems. For future work, we will extend the proposed techniques to deal with platforms and task systems with shared resources and task synchronization.

References

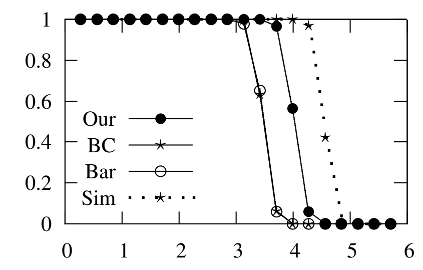
[1] B. Andersson and J. Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. *Technical Report, Chalmers University of Technology.*, 2001.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. In *Software Engineering Journal*, 1993.

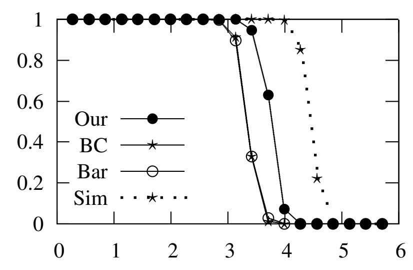
[3] T. P. Baker and M. Cirinei. A unified analysis of global edf and fixed-task-priority schedulability of sporadic task systems on multiprocessors. In *Journal of Embedded Computing*, 2007.



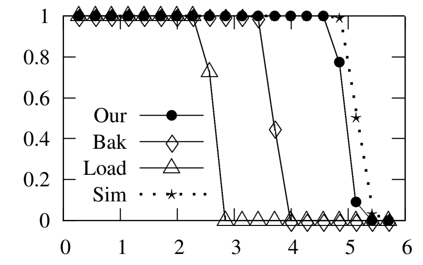
(a) $U_i \in [0.05, 0.1]$, $\frac{D_i}{T_i} \in [0.8, 1]$



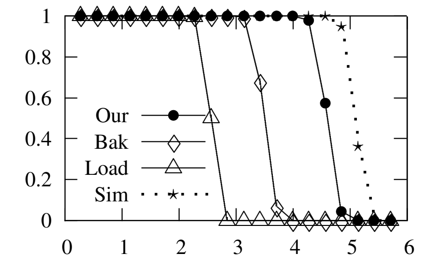
(b) $U_i \in [0.05, 0.15]$, $\frac{D_i}{T_i} \in [0.8, 1]$



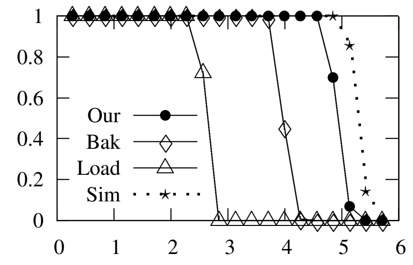
(c) $U_i \in [0.05, 0.2]$, $\frac{D_i}{T_i} \in [0.8, 1]$



(d) $U_i \in [0.05, 0.1]$, $\frac{D_i}{T_i} \in [0.8, 2]$



(e) $U_i \in [0.05, 0.2]$, $\frac{D_i}{T_i} \in [0.8, 2]$



(f) $U_i \in [0.05, 0.2]$, $\frac{D_i}{T_i} \in [0.8, 4]$

Figure 6. Acceptance Ratio: X-axis is total utilization $\sum_i U_i$; Y-axis is acceptance ratio.

[4] Theodore P. Baker. A comparison of global and partitioned edf schedulability tests for multiprocessors. In *Technical Report, Department of Computer Science, Florida State University, FL*, 2005.

[5] Sanjoy Baruah and Nathan Fisher. Global deadline-monotonic scheduling of arbitrary-deadline sporadic task systems. In *OPODIS*, 2007.

[6] Sanjoy K. Baruah. Techniques for multiprocessor global schedulability analysis. In *RTSS*, 2007.

[7] Sanjoy K. Baruah and Nathan Fisher. Global fixed-priority scheduling of arbitrary-deadline sporadic task system. In *ICDCN*, 2008.

[8] Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *RTSS*, 2007.

[9] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *OPODIS*, 2005.

[10] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. In *IEEE Transactions on Parallel and Distributed Systems*, 2008.

[11] Enrico Bini, Thi Huyen Châu Nguyen, Pascal Richard, and Sanjoy K. Baruah. A response-time bound in fixed-priority scheduling with arbitrary deadlines. *IEEE Trans. Comput.*, 58(2):279–286, 2009.

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

[13] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson. Litmusrt: A testbed for empirically comparing real-time multiprocessor schedulers. In *RTSS*, 2006.

[14] Nan Guan, Wang Yi, Zonghua Gu, Qingxu Deng, and Ge Yu. New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms. In *RTSS*, 2008.

[15] M. Gonzalez Harbour J. Palencia Gutierrez. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, 1998.

[16] M. Joseph and P.K. Pandya. Finding response times in a real-time system. In *The Computer Journal*, 1986.

[17] John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS*, 1990.

[18] L. Lundberg. Multiprocessor scheduling of age constraint processes. In *RTCSA*, 1998.

[19] On-Line Applications Research Corporation (OAR). Rtems applications c user's guide. 2001.

[20] K. Tindell, H. Hansson, and A. Wellings. Analysing realtime communications: Controller area network (can). In *RTSS*, 1994.

A APPENDIX

A.1 Proof of Lemma 3

Proof. We recall that by definition, $\Omega_k(x)$ consists of two sums over the sets τ^{NC} and τ^{CI} which are a partitioning of τ such that $\Omega_k(x)$ is maximal:

$$\Omega_k(x) = \sum_{\tau_i \in \tau^{\mathsf{CI}}} I_k^{\mathsf{CI}}(\tau_i, x, h) + \sum_{\tau_i \in \tau^{\mathsf{NC}}} I_k^{\mathsf{NC}}(\tau_i, x, h)$$

Let $\vartheta^{\mathsf{CI}} \subseteq \tau^{\mathsf{CI}}$ and $\vartheta^{\mathsf{NC}} \subseteq \tau^{\mathsf{NC}}$ be subsets of both partitions, such that:

$$\forall \tau_i \in \vartheta^{\mathsf{CI}} : W_k^{\mathsf{CI}}(\tau_i, x) > x - C_k + 1$$
$$\forall \tau_i \in \vartheta^{\mathsf{NC}} : W_k^{\mathsf{NC}}(\tau_i, x) > x - C_k + 1$$

Thus, $\vartheta := \vartheta^{\mathsf{CI}} \cup \vartheta^{\mathsf{NC}}$ captures the relatively "dense" tasks of τ. Using this, $I_k^{\mathsf{CI}}(\tau_i, x, h)$ and $I_k^{\mathsf{NC}}(\tau_i, x, h)$ can be rewritten using $W_k^{\mathsf{CI}}(\tau_i, x)$ and $W_k^{\mathsf{NC}}(\tau_i, x)$ in the definition of $\Omega_k(x)$:

$$\Omega_k(x) = |\vartheta| \cdot (x - C_i + 1) + \sum_{\tau_i \in \tau^{\mathsf{CI}} \setminus \vartheta^{\mathsf{CI}}} W_k^{\mathsf{CI}}(\tau_i, x) + \sum_{\tau_i \in \tau^{\mathsf{NC}} \setminus \vartheta^{\mathsf{NC}}} W_k^{\mathsf{NC}}(\tau_i, x) \tag{22}$$

We consider the case of $|\vartheta| < M$. (Otherwise, the lemma obviously holds.)

Now let $x < f_k - t_0$, as in the assumption of the lemma, so the Job J_k is still active at time point x. Thus, only at strictly less than C_k time points of the interval $[t_0, t_0 + x)$, J_k was able to run. Now we know, that all tasks from ϑ could keep at most $|\vartheta|$ processors busy at each time unit during the interval. It follows, that the remaining tasks (those from $\tau \setminus \vartheta$) kept the remaining $M - |\vartheta|$ processors busy for at least $x - C_k + 1$ time units during the interval (otherwise, J_k would have been able to execute for C_k time units and thus finish until $t_0 + x$). Consequently, the tasks from $\tau \setminus \vartheta$ must have generated a workload of at least $(M - |\vartheta|) \cdot (x - C_k + 1)$ over the considered x time units. Since $W_k^{\mathsf{CI}}(\tau_i, x)$ and $W_k^{\mathsf{NC}}(\tau_i, x)$ are upper bounds of their workloads, we have:

$$\sum_{\tau_i \in \tau^{\mathsf{CI}} \setminus \vartheta^{\mathsf{CI}}} W_k^{\mathsf{CI}}(\tau_i, x) + \sum_{\tau_i \in \tau^{\mathsf{NC}} \setminus \vartheta^{\mathsf{NC}}} W_k^{\mathsf{NC}}(\tau_i, x) \geq (M - |\vartheta|) \cdot (x - C_k + 1) \tag{23}$$

From (22) and (23) it follows

$$\Omega_k(x) \geq M \cdot (x - C_k + 1),$$

which is equivalent to the lemma. □

A.2 Proof of Theorem 3

Proof. We do a case distinction for the inequality, concerning the relation between left-hand side (LHS) and right-hand side (RHS).

First Case "LHS $<$ RHS": We assume

$$\sum_{i<k} V_i^k + M \cdot U_k < M \tag{24}$$

and want to show that there exists $h \geq 1$ such that at least one of the conditions $\mathsf{Term}(h,k)$ and $\mathsf{Miss}(h,k)$ holds. Let's assume that $\mathsf{Miss}(h,k)$ does not hold for any $h \geq 1$, which implies that (19) always has a minimal solution.

Let's also assume that $\mathsf{Term}(h,k)$ never holds, i.e. $\forall h \geq 1 : \chi^h > h \cdot T_k$, and let's pick an arbitrary such h. Thus, we know:

$$\epsilon := \chi^h - h \cdot T_k > 0. \tag{25}$$

We will now first bound $\Omega_k(\chi^h, h)$ from above and then use this to bound χ^h. Since h was chosen arbitrarily and $(\chi^h)_{h\geq 1}$ is a strictly increasing sequence, this will be the sufficient contradiction.

From the definition of $\Omega_k(x,h)$, we can derive:

$$\Omega_k(\chi^h, h) \leq \sum_{i<k} I_k^{\mathsf{CI}}(\tau_i, \chi^h, h)$$
$$\iff \Omega_k(\chi^h, h) \leq \sum_{i<k} \min(W_k^{\mathsf{CI}}(\tau_i, \chi^h), \chi^h - h \cdot C_k + 1)$$

From the definition of ϵ we can derive $h \cdot C_k = (\chi^h - \epsilon) \cdot U_k$. Using both that and $W_k^{\mathsf{CI}}(\tau_i, x) \leq x \cdot U_i + 2 \cdot C_i$, we get:

$$\Omega_k(\chi^h, h) \leq \sum_{i<k} \min(\chi^h \cdot U_i + 2 \cdot C_i, \chi^h - (\chi^h - \epsilon) \cdot U_k + 1)$$
$$\iff \Omega_k(\chi^h, h) \leq \sum_{i<k} \min(\chi^h \cdot U_i + 2 \cdot C_i, \chi^h \cdot (1 - U_k) + \epsilon \cdot U_k + 1)$$

We use the property $\min(a+b, c) \leq \min(b,c) + a$, and get:

$$\Omega_k(\chi^h, h) \leq \sum_{i<k} \min(\chi^h \cdot U_i, \chi^h \cdot (1 - U_k) + \epsilon \cdot U_k + 1) + 2 \cdot \sum_{i<k} C_i$$

Further, it holds $\min(a+b, c) \leq \min(b,c)$ if $b \geq c$. Using that, and setting η to be the number of tasks satisfying $U_i > 1 - U_k$, we get:

$$\Omega_k(\chi^h, h) \leq \sum_{i<k} \min(\chi^h \cdot U_i, \chi^h \cdot (1 - U_k)) + 2 \cdot \sum_{i<k} C_i + (\epsilon \cdot U_k + 1) \cdot \eta$$

From the initial assumption (24), we know $\eta \leq M - 1$, so we get:

$$\Omega_k(\chi^h, h) \leq \chi^h \sum_{i<k} \min(U_i, 1 - U_k) + 2 \cdot \sum_{i<k} C_i + (\epsilon \cdot U_k + 1) \cdot (M - 1)$$

Finally, using that $\epsilon > 0$, we get the upper bound on $\Omega_k(\chi^h, h)$:

$$\Omega_k(\chi^h, h) < \chi^h \sum_{i<k} V_i^k + 2 \cdot \sum_{i<k} C_i + (\epsilon \cdot U_k + 1) \cdot M \tag{26}$$

Now, to turn this into an upper bound for χ^h, we use that χ^h is a solution of the recursive Equation (19), i.e., it holds that:

$$\chi^h = \left\lfloor \frac{\Omega_k(\chi^h, h)}{M} \right\rfloor + h \cdot C_k \leq \frac{\Omega_k(\chi^h, h)}{M} + h \cdot C_k$$

Applying (26) to that, we get:

$$\chi^h < \frac{\chi^h \sum_{i<k} V_i^k + 2 \cdot \sum_{i<k} C_i + (\epsilon \cdot U_k + 1) \cdot M}{M} + h \cdot C_k$$

We now recall $\epsilon \cdot U_k = \chi^h \cdot U_k - h \cdot C_k$, thus:

$$\chi^h < \frac{\chi^h \sum_{i<k} V_i^k + 2 \cdot \sum_{i<k} C_i + (\chi^h \cdot U_k - h \cdot C_k + 1) \cdot M}{M} + h \cdot C_k$$

We simplify the inequality such that χ^h only appears on one side:

$$\chi^h \cdot (M - \sum_{i<k} V_i^k - M \cdot U_k) < 2 \cdot \sum_{i<k} C_i + M$$

From assumption (24) we know that $M - \sum_{i<k} V_i^k - M \cdot U_k > 0$, so we finally get:

$$\chi^h < \frac{2 \cdot \sum_{i<k} C_i + M}{M - \sum_{i<k} V_i^k - M \cdot U_k}$$

Second Case "LHS $>$ RHS": We now assume

$$\sum_{i<k} V_i^k + M \cdot U_k > M \tag{27}$$

and want to show that there exists $h \geq 1$ such that at least one of the conditions $\mathsf{Term}(h,k)$ and $\mathsf{Miss}(h,k)$ holds, i.e., that $\chi^h \leq h \cdot T_k$ or $\chi^h > (h-1) \cdot T_k + D_k$. (Note that this is trivially true for $D_k \leq T_k$, so we focus on the $D_k > T_k$ case.)

Let's assume on the contrary, that neither of them holds for any $h \geq 1$, i.e., that

$$\forall h \geq 1 : \chi^h \in \big(h \cdot T_k, (h-1) \cdot T_k + D_k\big). \tag{28}$$

We will now first bound $\Omega_k(\chi^h, h)$ from below and then use this to bound χ^h from above. Since h was chosen arbitrarily and $(\chi^h)_{h\geq 1}$ is a strictly increasing sequence, this will be the sufficient contradiction.

From the definition of $\Omega_k(x,h)$, we can derive:

$$\Omega_k(\chi^h, h) \geq \sum_{i<k} I_k^{\mathsf{NC}}(\tau_i, \chi^h, h)$$
$$\iff \Omega_k(\chi^h, h) \geq \sum_{i<k} \min(W_k^{\mathsf{NC}}(\tau_i, \chi^h), \chi^h - h \cdot C_k + 1)$$

Using $W_k^{\mathsf{NC}}(\tau_i, x) \geq x \cdot U_i$, we get:

$$\Omega_k(\chi^h, h) \geq \sum_{i<k} \min(\chi^h \cdot U_i, \chi^h - h \cdot C_k + 1)$$

From the assumption (28) we further know $\chi^h > h \cdot T_k$, from which we can derive:

$$\chi^h - h \cdot C_k + 1 > \chi^h(1 - U_k) + 1$$

Using that in the above, we get:

$$\Omega_k(\chi^h, h) \geq \sum_{i<k} \min(\chi^h \cdot U_i, \chi^h(1 - U_k))$$

Finally, cleaning up using the definition of V_i^k:

$$\Omega_k(\chi^h, h) \geq \chi^h \sum_{i<k} V_i^k \tag{29}$$

Now, to turn this into an upper bound for χ^h, we use that χ^h is a solution of Equation (19), i.e., it holds that:

$$\chi^h = \left\lfloor \frac{\Omega_k(\chi^h, h)}{M} \right\rfloor + h \cdot C_k > \frac{\Omega_k(\chi^h, h)}{M} + h \cdot C_k - 1$$

Applying (29) to that, we get:

$$\chi^h > \frac{\chi^h \sum_{i<k} V_i^k}{M} + h \cdot C_k - 1$$

Or, equivalently:

$$\chi^h \sum_{i<k} V_i^k + M \cdot h \cdot C_k - M < M \cdot \chi^h$$

From assumption (28), we further know that $\chi^h < (h-1)T_k + D_k$, so we have $1 + (\chi^h - D_k)/T_k < h$. We apply that:

$$\chi^h \sum_{i<k} V_i^k + M \cdot (1 + (\chi^h - D_k)/T_k) \cdot C_k - M < M \cdot \chi^h$$

After ordering the operands, we have:

$$\chi^h (\sum_{i<k} V_i^k + M \cdot U_k - M) < M(1 + D_k \cdot U_k - C_k)$$

Finally, assumption (27) allows us to get an upper bound for χ^h:

$$\chi^h < \frac{M(1 + D_k \cdot U_k - C_k)}{\sum_{i<k} V_i^k + M \cdot U_k - M}$$

□