Techniques for Multiprocessor Global Schedulability Analysis*

Sanjoy Baruah
The University of North Carolina at Chapel Hill

Abstract

The scheduling of sporadic task systems upon multiprocessor platforms is considered, when inter-processor migration is permitted. It is known that current schedulability tests for such systems perform quite poorly when compared to schedulability tests for partitioned scheduling. Limitations of current tests are identified, which may be responsible for the unsatisfactory performance of these tests. A new test that overcomes some of these limitations is proposed and proved correct.

1 Introduction and Motivation

A real-time system is often modelled as a finite collection of independent recurring tasks, each of which generates a potentially infinite sequence of *jobs*. Every job is characterized by an arrival time, an execution requirement, and a deadline, and it is required that a job complete execution between its arrival time and its deadline. Different formal models for recurring tasks place different restrictions on the values of the parameters of jobs generated by each task. One of the more commonly used formal models is the *sporadic task model* [23, 9], which will be considered in this paper.

Scheduling is the allocation of processor time to jobs, and a *scheduling algorithm* is used for determining such allocation. A *schedulability test* for a given scheduling algorithm accepts as input the specifications of a real-time system, and determines whether the scheduling algorithm can guarantee to schedule the system such that all jobs of all tasks will meet all deadlines, under all permissible combinations of job-arrival sequences by the different tasks comprising the system. In this paper, we study the scheduling of systems of sporadic tasks upon a platform comprised of several identical processors. In scheduling a task system upon such a platform, it is possible to divide the processors into clusters and assign each task to a cluster. A scheduling algorithm is then applied locally in each cluster, to the jobs generated by the tasks that are assigned to the cluster. It is assumed that inter-processor communication within a cluster incurs no overhead. We focus here on the two extremes of clustering. In *partitioned scheduling*, each processor is a cluster of size one. That is, each task is assigned to one processor and all the jobs generated by a task are constrained to execute only upon the processor to which the task has been assigned. In *global scheduling*, by contrast, there is only one cluster containing all the processors. A job may execute upon any processor, and a preempted job may later resume execution upon the same processor as, or a different processor from, the one on which it had previously been executing. However, each job may execute on at most one processor at each instant in time.

Current state of the art. Currently, the partitioned scheduling of sporadic task systems is much better understood than global scheduling. Sufficient schedulability tests of polynomial time-complexity have been designed [7, 17, 8] for various commonly-used scheduling algorithms (such as Earliest Deadline First (EDF) [22, 14] and Deadline Monotonic (DM) [21]). Worst-case resource-augmentation bounds, that provide a quantitative measure of how effective these tests are, have been obtained. These schedulability tests have also been extensively evaluated via simulations [4, 6], and shown to have much better average-case behavior than indicated by their worst-case guarantees. In contrast, we are not aware of non-trivial theoretical bounds on the performance of known sufficient schedulability tests [2, 3, 10] for global scheduling (other than a relatively naive resource-augmentation bound on global DM [16]), and simulation experiments have tended to indicate that they perform poorly in comparison to the partitioned schedulability tests.

This research. This research is aimed at obtaining a better understanding of global schedulability for sporadic task systems. After formally defining the task and machine models in Section 2, we start out in Section 3 by highlighting what appears to be one of the root causes of difficulty in such analysis: our failure thus far to come up with tech-

*Supported in part by NSF Grant Nos. CNS-0408996, CCF-0541056, and CCR-0615197, ARO Grant No. W911NF-06-1-0425, and funding from the Intel Corporation.

1052-8725/07 $25.00 © 2007 IEEE
DOI 10.1109/RTSS.2007.35

niques for characterizing the "worst case" behavior of such systems. In Section 4, we demonstrate that global and partitioned scheduling are *incomparable* for priority-based scheduling algorithms that are not allowed to dynamically change the priorities of jobs; this extends a result of Leung and Whitehead [21], who obtained a similar incomparability result for algorithms in which all jobs of each task are required to have the same priority. In Section 5, we briefly summarize prior tests for global EDF-schedulability analysis, and highlight their features and disadvantages. In Section 6, we derive and analyze a new global-EDF schedulability test that overcomes some of the disadvantages of these prior tests. We conclude in Section 7 with a summary of the main results in this paper.

For the sake of concreteness, we have chosen to focus in this paper upon a specific global scheduling algorithm – EDF. However, the issues are similar for many other scheduling algorithms, and many of the techniques that are applicable to global EDF scheduling are likely to be applicable to these other scheduling algorithms as well. (For example, Baker has used similar techniques to study both global EDF-schedulability [2, 3] and global DM-schedulability [2, 5], and Bertogna et al. have extended their global EDF-schedulability results [10] to apply to DM scheduling [11].)

2 Model

A *sporadic task* $\tau_i = (C_i, D_i, T_i)$ is characterized by a *worst-case execution requirement* C_i, a *(relative) deadline* D_i, and a *minimum inter-arrival separation* T_i, which is, for historical reasons, also referred to as the *period* of the task. Such a sporadic task generates a potentially infinite sequence of jobs, with successive job-arrivals separated by at least T_i time units. Each job has a worst-case execution requirement equal to C_i and a deadline that occurs D_i time units after its arrival time. We refer to the interval, of size D_i, between such a job's arrival instant and deadline as its *scheduling window.* We assume a fully *preemptive* execution model: any executing job may be interrupted at any instant in time, and its execution resumed later with no cost or penalty. A *sporadic task system* is comprised of several such sporadic tasks. Let τ denote a system of such sporadic tasks: $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$, with $\tau_i = (C_i, D_i, T_i)$ for all i, $1 \leq i \leq n$. Task system τ is said to be a *constrained* sporadic task system if it is guaranteed that each task $\tau_i \in \tau$ has its relative deadline parameter no larger than its period: $D_i \leq T_i$, and an *implicit-deadline* sporadic task system if $D_i = T_i$ for all $\tau_i \in \tau$. (Implicit-deadline systems are also known as *Liu and Layland* [22] task systems.) ***In this paper, we restrict our attention to constrained and implicit-deadline task systems.***

Schedulability. A real-time system implemented on a particular computing platform is said to be A-schedulable with respect to a given scheduling algorithm A, if the algorithm A schedules the system such that all jobs of all tasks will meet all deadlines, under all permissible (also called *legal*) combinations of job-arrival sequences by the different tasks comprising the system. A *schedulability test* for scheduling algorithm A (also called an A*-schedulability test*) accepts as input the specifications of a real-time system, and determines whether the system is A-schedulable or not. An A-schedulability test is said to be *exact* if it correctly identifies all A-schedulable systems, and *sufficient* if it may fail to identify some A-schedulable systems (it must guarantee, though, that all identified systems are indeed A-schedulable.)

Task and system characteristics. The concepts of task and system *utilization* and *density* prove useful in the analysis of sporadic task systems on multiprocessors. These concepts are defined as follows:

Utilization: The utilization u_i of a task τ_i is the ratio C_i/T_i of its execution time to its period. The total utilization $u_{\rm sum}(\tau)$ and the largest utilization $u_{\rm max}(\tau)$ of a task system τ are defined as follows:

$$u_{\rm sum}(\tau) \stackrel{\rm def}{=} \sum_{\tau_i \in \tau} u_i; \qquad u_{\rm max}(\tau) \stackrel{\rm def}{=} \max_{\tau_i \in \tau}(u_i)$$

Density: The density δ_i of a task τ_i is the ratio C_i/D_i of its execution time to its relative deadline. The total density $\delta_{\rm sum}(\tau)$ and the largest density $\delta_{\rm max}(\tau)$ of a task system τ are defined as follows:

$$\delta_{\rm sum}(\tau) \stackrel{\rm def}{=} \sum_{\tau_i \in \tau} \delta_i; \qquad \delta_{\rm max}(\tau) \stackrel{\rm def}{=} \max_{\tau_i \in \tau}(\delta_i)$$

An additional concept that plays a critical role in the schedulability analysis of sporadic task systems is that of **demand bound function**. For any interval length t, the demand bound function $\mathrm{DBF}(\tau_i, t)$ of a sporadic task τ_i bounds the maximum cumulative execution requirement by jobs of τ_i that both arrive in, and have deadlines within, any interval of length t. It has been shown [9] that

$$\mathrm{DBF}(\tau_i, t) = \max\left(0, \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i\right)$$

A **load** parameter, based upon the DBF function, may be defined for any sporadic task system τ as follows:

$$\mathrm{LOAD}(\tau) \stackrel{\rm def}{=} \max_{t>0}\left(\frac{\sum_{\tau_i \in \tau} \mathrm{DBF}(\tau_i, t)}{t}\right)$$

Fixed job-priority (FJP) scheduling. Run-time scheduling algorithms are typically implemented as follows: at each time instant, a *priority* is assigned to each job that has been released but not yet completed, and the available processors are allocated to the highest-priority jobs. In EDF scheduling, jobs are assigned priorities according to their deadline parameters: the earlier the deadline of a job, the greater its priority.

It has long been known that global EDF, in common with many other global scheduling schemes, suffers from the so-called Dhall effect [15] which results in task systems with arbitrarily low utilization and density not being EDF-schedulable. In order to circumvent the Dhall effect, hybrid variants of EDF have been designed [18] in which all the jobs of some selected tasks are assigned highest priority, and the remaining tasks' jobs are assigned priorities according to EDF. Such priority-driven scheduling algorithms are instances of a class of algorithms called *fixed job-priority* (FJP) scheduling algorithms. In FJP scheduling algorithms, the priority of a job may not change between its arrival time and the instant it completes execution; however, different jobs of the same task may have different priorities. (In contrast, fixed task-priority (FTP) algorithms require that all jobs of a task have the same –fixed– priority, while in dynamic-priority (DP) algorithms the priority of a job may change between its arrival time and its completion time. It is evident from these definitions that FJP scheduling is a generalization of FTP scheduling, and DP scheduling is a generalization of FJP scheduling.)

3 Global scheduling and Worst-case behavior

Recall that in order for a sporadic task system to be considered A-schedulable for a given scheduling algorithm A, A must generate schedules meeting all deadlines for all legal job-arrival sequences of the task system. Since any sporadic task system has infinitely many different legal job arrival sequences, an approach of exhaustive enumeration of such sequences, followed by simulating algorithm A on each of these sequences, will not yield an effective schedulability test. Instead, the typical approach has been to identify one or a few *worst-case job arrival sequences* for a task system for which it can be proved that if algorithm A successfully schedules all these worst-case job arrival sequences, then A is guaranteed to successfully schedule all legal job arrival sequences of the task system. For example, with respect to EDF or Deadline-Monotonic (DM) scheduling upon preemptive uniprocessors it is known [22, 20, 9] that there is a unique worst-case job arrival sequence: every task has one job arrive at the same instant in time, and each task has subsequent jobs arrive as soon as legally permitted to do so. (Such a job arrival sequence is often called the *synchronous arrival sequence* for the task system.) It has been shown [7, 17, 8] that the synchronous arrival sequence is also the worst-case job arrival sequence for partitioned multiprocessor EDF and DM scheduling.

For global scheduling, however, the synchronous arrival sequence is not necessarily the worst-case arrival sequence, as the following example illustrates with respect to EDF scheduling.

Example 1 Consider the task system comprised of the three tasks $\tau_1 = (1, 1, 2)$, $\tau_2 = (1, 1, 3)$, and $\tau_3 = (5, 6, 6)$, executing upon two unit-capacity processors. It may be seen (Figure 1 (a)) that the synchronous arrival sequence is successfully scheduled by EDF to meet all deadlines. However, if task τ_1's second job were to arrive three, rather than two, time units after the first (Figure 1 (b)), then EDF would miss some deadline over the interval $[0, 6]$ (and indeed, no scheduling algorithm can possibly guarantee to meet all deadlines for this job arrival sequence). ■

Without knowing what the worst-case behavior of a sporadic task system may be, it is not possible to design simulation-based exact (necessary as well as sufficient) schedulability tests. To our knowledge, no finite collection of worst-case job arrival sequences has been identified for global scheduling of sporadic task systems. This, at a basic level, is the fundamental difference between our understanding of partitioned and global scheduling of sporadic task systems. As stated above, it is known that the synchronous arrival sequence represents the worst-case behavior of a task system under partitioned scheduling. Hence, partitioned schedulability testing can be solved in principle by determining whether it is possible to partition the tasks among the processors such that no deadlines are missed in the synchronous arrival sequence. An exact algorithm for solving this is provably highly intractable, but approximation algorithms that run in polynomial time and exhibit excellent average-case and provably bounded worst-case behavior have been devised [7, 17, 8].

In the global case, however, we do not know how to determine schedulability even if computational tractability were not an issue. That is, we do not yet have an adequate understanding of what precisely the characteristics of a globally schedulable system are; without knowing these characteristics, it seems futile to seek accurate, efficient tests for identifying such systems. (In this regard, Example 1 above illustrates that LOAD is not an exact indicator of schedulability: while $\text{LOAD}(\tau) \leq m$ is clearly necessary for τ to be EDF-schedulable on an m-processor platform, the task system in Example 1 illustrates that this is not sufficient.)

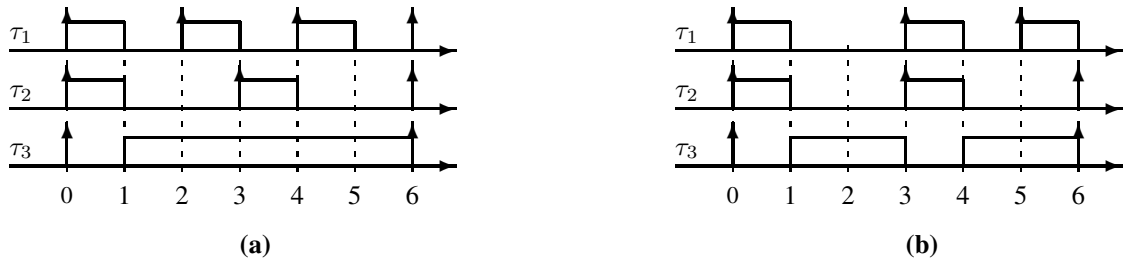

Figure 1. Figure for Example 1, illustrating that the synchronous arrival sequence does not represent the worst case.

4 Partitioned versus global FJP scheduling

As stated in Section 1 above, simulation experiments [4, 6] have shown that currently-known global schedulability tests perform poorly compared to partitioned schedulability tests. That is, it appears that a far greater fraction of randomly-generated sporadic task systems are deemed schedulable by currently-known partitioned schedulability tests than by currently-known global schedulability tests.

One might consider this result surprising, since at first glance it appears that partitioned scheduling (which forbids jobs from migrating between processors) is a special case of global scheduling (which allows, but does not require, such migration). However, this is erroneous. It has previously been shown by Leung and Whitehead [21, page 242] with regard to fixed task priority (FTP) scheduling schemes that there are sporadic task systems schedulable by some global algorithm but not by any partitioned algorithm, and that there are other sporadic task systems that are schedulable by some partitioned algorithm but not by any global algorithm. Theorem 1 below demonstrates similar incomparability between global and partitioned scheduling for the more general class of FJP scheduling algorithms as well.

As one might expect, there are systems that global algorithms can schedule but partitioned ones cannot; this is formally shown in Lemma 1 below. Somewhat counter-intuitively, it turns out (Lemma 2 below) that there are systems which partitioned FJP algorithms can schedule, which cannot be scheduled by any global FJP algorithm.

Lemma 1 *There are task systems that are schedulable using global FJP algorithms that partitioned FJP algorithms cannot schedule.*

Proof Sketch: Here's an example task system, on 2 processors:

$$\tau_1 = (2, 2, 3), \tau_2 = (3, 3, 4), \tau_3 = (5, 12, 12)$$

Clearly, this system cannot be scheduled using partitioning, since $u_1 + u_2$, $u_1 + u_3$, and $u_2 + u_3$ are each > 1. To show that a global FJP algorithm can schedule this task system, consider any FJP algorithm which assigns lowest priority to τ_3's jobs. Since there are two processors, τ_1 and τ_2's jobs are guaranteed to meet all deadlines. Now observe that each job of τ_3 needs to execute for 5 time-slots over an interval of 12 slots[1]. Now over any 12 contiguous slots, τ_1 may execute during at most 8 slots, thereby leaving at least 4 slots for τ_3's job upon one processor.

- If τ_1 executes for fewer than 8 slots between a job of τ_3's scheduling window, then it is straightforward to see that the job of τ_3 completes execution by its deadline, since no more than one other job (of τ_2) will be active during the at least 5 time units of this job's scheduling window.

- If τ_1's jobs execute for exactly 8 slots during the scheduling window of one of τ_3's jobs, it is easily seen that τ_1's jobs must be arriving exactly 3 time-units apart. Hence, it is not possible that τ_3's jobs execute in parallel with τ_1's jobs during all 8 of the time-slots during which τ_1's jobs execute within this scheduling window. Therefore there are at least 5 slots within this scheduling window during which one of both processors is idle after both τ_1 and τ_2 have been scheduled.

■

Perhaps surprisingly, the converse of the above lemma is also true:

Lemma 2 *There are task systems that are schedulable using partitioned FJP algorithms that global FJP algorithms cannot schedule.*

Proof Sketch: Here's an example task system, on 2 processors:

$$\tau_1 = (2, 2, 3), \tau_2 = (3, 3, 4), \tau_3 = (4, 12, 12), \tau_4 = (3, 12, 12)$$

This task system may be partitioned by assigning τ_1 and τ_3 to one processor and the other two tasks to the remaining

[1] For ease of exposition, we are assuming a slotted model of time for this proof sketch; however, it is easily generalized to a non-slotted model.

processor, and scheduling each processor using uniprocessor EDF.

To show that no global FJP priority assignment scheme can meet all deadlines, consider the synchronous arrival sequence over the interval $[0, 12)$. It may be verified that whichever of τ_3's and τ_4's job has lower priority ends up missing its deadline while one processor goes idle over $[11, 12)$. ■

Hence our intuition that global is a generalization of partitioned turns out to be incorrect for FJP scheduling as well as FTP scheduling:

Theorem 1 *Global and partitioned FJP scheduling are **incomparable**.*

One implication of this is that the current state of the art — partitioned schedulability tests being far superior to global ones — are not contradicted by facts (only intuition). That is, it has not been ruled out that a far larger fraction of sporadic task systems are schedulable under partitioned FJP scheduling as compared to global FJP scheduling.

5 Global EDF schedulability: Prior results

In this section, we provide brief summaries of currently-known tests for global EDF-schedulability analysis of sporadic task systems. In addition, we highlight salient features and drawbacks of these tests; efforts to overcome these drawbacks will drive the design of the new test we propose in the next section.

The density, [BAK], and [BCL] tests. The global multiprocessor scheduling of implicit-deadline (Liu and Layland) sporadic task systems was studied in [18]. It was shown that $[u_{\text{sum}}(\tau) \leq m - (m-1)u_{\max}(\tau)]$ is a sufficient condition for implicit-deadline sporadic task system τ to be global EDF-schedulable upon m unit-capacity processors. Minor extensions to the proofs in [18] can be used to obtain the following sufficient global-EDF schedulability test for constrained-deadline sporadic task systems:

$$\delta_{\text{sum}}(\tau) \leq m - (m-1)\delta_{\max}(\tau) \; . \tag{1}$$

This test is often referred to in the literature as the *density* test for global EDF-schedulability of sporadic task systems.

Meanwhile Baker [2, 3, 10] designed a test, based on some deep insights, for global-EDF schedulability analysis. In essence, Baker's test – henceforth referred to here as the [BAK] test – is obtained by assuming that a task τ_k's job misses its deadline, and then determining necessary conditions on the parameters of all the tasks that must be satisfied in order for such a deadline miss to occur. Then *negating* these conditions for each τ_k yields a sufficient test for global-EDF schedulability.

In 2005, Bertogna et al. [10] used ideas very similar to Baker's to came up with a simpler test that occasionally outperformed both the density and the [BAK] tests. The test from [10] will be referred to here as the [BCL] test.

Overview of the [BAK] and [BCL] tests. Both tests are built around the following general strategy, first introduced by Baker [2]. Suppose that τ is not global EDF-schedulable, and consider a legal sequence of job requests of τ on which global EDF misses one or more deadlines. Suppose that a job of τ_k is the first job that misses its deadline, at some time-instant t_d. Both tests analyze the situation over some interval $[t_o, t_d)$, where t_o is different in the [BAK] and [BCL] tests (and is precisely defined for each test). Both tests compute upper bounds on the amount of work that EDF can be required to execute over the interval $[t_o, t_d)$, and obtain an unschedulability condition by setting this bound to be large enough to deny τ_k's job C_k units of execution over its scheduling window.

In bounding the total amount of work that each task τ_i needs to have executed over $[t_o, t_d)$ in the EDF schedule, one must consider **(i)** jobs of τ_i that arrive within the interval, *and* **(ii)** possibly one additional job that arrives prior to t_o but has not completed execution by time-instant t_o and hence "carries in" some execution into the interval. The contribution of jobs arriving within the interval may be computed using the demand bound abstraction (Section 2 above), but new techniques are needed for bounding the carry-in. Both the [BAK] and [BCL] tests use different bounds on the amount of such work for each τ_i.

Differences between the [BAK] and [BCL] tests. Despite the similarities in overall approach, the two tests differ quite significantly in the details:

1 In [BCL], t_o is set equal to $(t_d - D_k)$; i.e., the arrival time of the job that misses its deadline. In [BAK], it is set equal to the earliest time-instant prior to this job's arrival time, at which a certain condition[2] is satisfied.

2 In [BCL], fairly primitive techniques are used to bound the amount of work that τ_i may contribute to this interval, while [BAK] uses a far more sophisticated analysis. Specifically, while [BAK] is able to show that parts of "carry-in" jobs – jobs whose scheduling windows span t_o – must have completed execution before t_o, [BCL] pessimistically assumes that each carry-in job executed as late as possible, immediately before their deadlines and hence carried in as much work as possible.

3 The length of the interval $[t_o, t_d)$ could be larger in the [BAK] tests than in the [BCL] test. This is desir-

[2] It is not essential for us to know what this condition is, in order to understand the remainder of this paper. Please consult [3] for details.

able since any additive inaccuracy introduced in computing bounds on the amount of work that must be executed over $[t_o, t_d)$ gets amortized over a larger interval. (However, a closer examination of the [BAK] test indicates that this potential advantage is not exploited – a final step in the derivation of the [BAK] test lower-bounds the size of the interval $[t_o, t_d)$ by D_k, which is the value used by the [BCL] test as well.)

Comparison. Although the [BAK] test employs considerably more sophisticated analysis than the [BCL] test, it does not unequivocally outperform the [BCL] test. Baker has conducted extensive simulation experiments comparing the density, [BAK] and [BCL] tests. These simulations have demonstrated that all three tests are incomparable: there are task systems deemed EDF-schedulable by each test but not by the other two tests. They also seem to demonstrate that the density and [BCL] tests, taken together, are able to cover most of the task systems that are handled by the [BAK] test; in Baker's words (Email communication, February 2007) *it seems that many of the cases where my test does better than the* [BCL] *test are also cases where the density test works, and many of the cases where my test does better than the density test the* [BCL] *test also works.*

Shortcomings common to both tests. The [BAK] and [BCL] tests share certain shortcomings.

1. First, in analyzing a possible deadline miss for τ_k both tests consider a "worst-case" scenario in which it is assumed that *every* one of the n tasks in the system carries work into the interval that must be considered. In systems with a large number of tasks ($n \gg m$), this results in severe over-estimation of the cumulative carry-in. This is further aggravated by the feature of both tests – outlined above – of amortizing this carry-in over an interval of size D_k. Due perhaps to these facts, our observation has been that *these tests tend to perform poorly on "non-uniform" task systems*: they are likely to flag as being not schedulable an inordinately large fraction of task systems in which different tasks' parameters are of different orders of magnitude.

2. Second, both tests have run-time polynomial in the representation of the task system: the [BCL] test runs in $\mathcal{O}(n^2)$ time and the [BAK] test in $\mathcal{O}(n^3)$ time, where n is the number of tasks in the system. It may seem strange to refer to this low run-time complexity as a "shortcoming," but the fact is that, in real-time scheduling theory, *pseudo-polynomial* run-time complexity is usually considered satisfactory for schedulability analysis; this is a consequence of the fact that task parameters are typically not too large in such systems. (For example, commonly-used exact uniprocessor EDF [9, 24] and DM [19, 1] schedulability tests have pseudo-polynomial run-time.) It would be nice if the [BCL] and [BAK] tests could be rendered more accurate by being allowed to run for pseudo-polynomial time; however, we are not aware of any technique that enables such a trade-off. So this low run-time complexity is a "shortcoming" in the sense that it provides a benefit (polynomial run-time) that is not particularly needed, and which we cannot trade in for a more needed benefit (greater accuracy).

6 An improved schedulability algorithm

We have designed a new global EDF-schedulability test that overcomes some of these shortcomings of the [BAK] and [BCL] tests. Our test possesses the following features that distinguishes it from the [BAK] and [BCL] tests: *(i)* it runs in time pseudo-polynomial in the representation of the task system; *(ii)* it considers intervals $[t_o, t_d)$ that do not necessarily coincide with the scheduling window of the task τ_k being assumed to miss its deadline; *(iii)* it allows us to bound the number of tasks carrying in work into the interval $[t_o, t_d)$ at $(m-1)$, where m is the number of processors in the system. In contrast to the [BAK] and [BCL] tests, it will be shown (Corollary 1) that our test generalizes the known exact EDF-schedulability test on uniprocessors.

Our algorithm, like the [BCL] test, draws inspiration from the seminal work of Baker [2, 3] that yielded the [BAK] test, and follows the same general framework. We consider each task τ_k separately; when considering a specific τ_k, we identify sufficient conditions for ensuring that τ_k cannot miss any deadlines. To ensure that no deadlines are missed by any task in τ, these conditions must be checked for each of the n tasks $\tau_1, \tau_2, \ldots, \tau_n$.

Consider any legal sequence of job requests of task system τ, on which EDF misses a deadline. Suppose that a job of task τ_k is the one to first miss a deadline, and that this deadline miss occurs at time-instant t_d (see Figure 2). Let t_a denote this job's arrival time: $t_a = t_d - D_k$.

Discard from the legal sequence of job requests all jobs with deadline $> d_k$, and consider the EDF schedule of the remaining (legal) sequence of job requests. Since later-deadline jobs have no effect on the scheduling of earlier-deadline ones under preemptive EDF, it follows that a deadline miss of τ_k occurs at time-instant t_d (and this is the earliest deadline miss), in this new EDF schedule

Let t_o denote the latest time-instant $\leq t_a$ at which at least one processor is idled in this EDF schedule. Let $A_k \stackrel{\text{def}}{=} t_a - t_o$.

As in [2, 3], our goal now is to identify conditions necessary for a deadline miss to occur; i.e., for τ_k's job to execute for strictly less than C_k time units over $[t_a, t_d)$. In order for

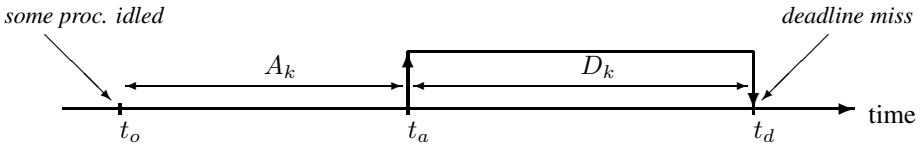

Figure 2. Notation. A job of task τ_k arrives at t_a and misses its deadline at time-instant t_d. The latest time-instant prior to t_a when not all m processors are busy is denoted t_o.

τ_k's job to execute for strictly less than C_k time units over $[t_a, t_d)$, it is necessary that all m processors be executing jobs other than τ_k's job for strictly more than $(D_k - C_k)$ time units over $[t_a, t_d)$. Let us denote by Γ_k a collection of intervals, not necessarily contiguous, of cumulative length $(D_k - C_k)$ over $[t_a, t_d)$, during which all m processors are executing jobs other than τ_k's job in this EDF schedule.

For each i, $1 \leq i \leq n$, let $I(\tau_i)$ denote the contribution of τ_i to the work done in this EDF schedule during $[t_o, t_a) \bigcup \Gamma_k$. In order for the deadline miss to occur, it is necessary that the total amount of work that executes over $[t_o, t_a) \bigcup \Gamma_k$ satisfy the following condition

$$\sum_{\tau_i \in \tau} I(\tau_i) > m \times (A_k + D_k - C_k)\ ; \qquad (2)$$

this follows from the observation that all m processors are, by definition, completely busy executing this work over the A_k time units in the interval $[t_o, t_a)$, as well as the intervals in Γ_k of total length $(D_k - C_k)$.

It is important to understand exactly what Equation 2 means. Equation 2 represents necessary conditions for task τ_k to miss a deadline A_k time units after an instant at which at least one processor is idled. *To show that a task system is* EDF*-schedulable, it therefore suffices to demonstrate that for all tasks τ_k and for all values of A_k, Equation 2 cannot be satisfied.*

Observe that the total length of the intervals in $[t_o, t_a) \bigcup \Gamma_k$ is equal to $(A_k + D_k - C_k)$.

Let us say that τ_i has a *carry-in job* in this EDF schedule if there is a job of τ_i that arrives before t_o and has not computed execution by t_o. In the following, we compute upper bounds on $I(\tau_i)$ if τ_i has no carry-in job (this is denoted as $I_1(\tau_i)$), or if it does (denoted as $I_2(\tau_i)$).

Computing $I_1(\tau_i)$. If a task τ_i contributes no carry-in work, then its contribution to this total amount of work that must execute over $[t_o, t_a) \bigcup \Gamma_k$ is generated by jobs arriving in, and having deadlines within, the interval $[t_o, t_d)$. Let us first consider $i \neq k$; in that case, it follows from the definition of the demand bound function (DBF — see Section 2 above) that the total work is at most $\text{DBF}(\tau_i, A_k + D_k)$; furthermore, this total contribution cannot exceed the total length of the intervals in $[t_o, t_a) \bigcup \Gamma_k$. Hence, the contribution of τ_i to the total work that must be done by EDF over $[t_o, t_a) \bigcup \Gamma_k$ is at most

$$\min(\text{DBF}(\tau_i, A_k + D_k), A_k + D_k - C_k).$$

Now, consider the case $i = k$. In that case, the job of τ_k arriving at time-instant t_a does not contribute to the work that must be done by EDF over $[t_o, t_a) \bigcup \Gamma_k$; hence, its execution requirement must be subtracted. Also, this contribution cannot exceed the length of the interval $[t_o, t_a)$; i.e., A_k.

Putting these pieces together, we get the following bound on the contribution of τ_i to the total work that must be done by EDF over $[t_o, t_a) \bigcup \Gamma_k$:

$$I_1(\tau_i) \stackrel{\text{def}}{=} \begin{cases} \min(\text{DBF}(\tau_i, A_k + D_k), A_k + D_k - C_k) & \text{if } i \neq k \\ \min(\text{DBF}(\tau_i, A_k + D_k) - C_k, A_k) & \text{if } i = k \end{cases} \qquad (3)$$

Computing $I_2(\tau_i)$. Let us now consider the situation when τ_i is active at t_o, and hence potentially carries in some work. It was shown in [10] that the total work of τ_i in this case can be upper-bounded by considering the scenario in which some job of τ_i has a deadline at t_d, and all jobs of τ_i execute at the very end of their scheduling windows.

Let us denote as $\text{DBF}'(\tau_i, t)$ the amount of work that can be contributed by τ_i over a contiguous interval of length t, if some job of τ_i has its deadline at the very end of the interval and each job of τ_i executes during the C_i units immediately preceding its deadline. It is easily seen (see Figure 3) that there are exactly $\lfloor t/T_i \rfloor$ complete jobs of τ_i within this interval, and an amount $\min(D_i, t \bmod T_i)$ of the scheduling window of an additional – carry-in – job.

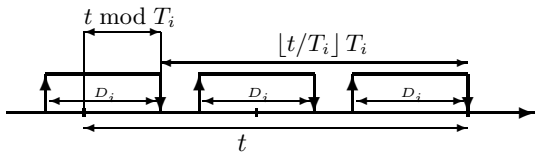

Figure 3. Computing $\text{DBF}'(\tau_i, t)$.

This carry-in scheduling window may bring in at most C_i units of execution, yielding the following expression for $\text{DBF}'(\tau_i, t)$:

$$\text{DBF'}(\tau_i, t) \stackrel{\text{def}}{=} \left\lfloor \frac{t}{T_i} \right\rfloor \times C_i + \min(C_i, t \bmod T_i) \quad (4)$$

In computing τ_i's contribution to the total amount of work that must execute over $[t_o, t_a) \bigcup \Gamma_k$, let us first consider $i \neq k$. In that case, it follows from the definition of the demand bound function (DBF', as defined above) that the total work is at most $\text{DBF}'(\tau_i, A_k + D_k)$; furthermore, this total contribution cannot exceed the total length of the intervals in $[t_o, t_a) \bigcup \Gamma_k$. Hence, the contribution of τ_i to the total work that must be done by EDF over $[t_o, t_a) \bigcup \Gamma_k$ is at most

$$\min(\text{DBF}'(\tau_i, A_k + D_k), A_k + D_k - C_k).$$

Now, consider the case $i = k$. In that case, the job of τ_k arriving at time-instant t_a does not contribute to the work that must be done by EDF over $[t_o, t_a) \bigcup \Gamma_k$; hence, its execution requirement must be subtracted. Also, this contribution cannot exceed the length of the interval $[t_o, t_a)$; i.e., A_k.

From the discussion above, we get the following bound on the contribution of τ_i to the total work that must be done by EDF over $[t_o, t_a) \bigcup \Gamma_k$:

$$I_2(\tau_i) \stackrel{\text{def}}{=} \begin{cases} \min(\text{DBF'}(\tau_i, A_k + D_k), A_k + D_k - C_k) & \text{if } i \neq k \\ \min(\text{DBF'}(\tau_i, A_k + D_k) - C_k, A_k) & \text{if } i = k \end{cases} \quad (5)$$

Putting the pieces together. Let us denote by $I_{\text{DIFF}}(\tau_i)$ the difference between $I_2(\tau_i)$ and $I_1(\tau_i)$:

$$I_{\text{DIFF}}(\tau_i) \stackrel{\text{def}}{=} I_2(\tau_i) - I_1(\tau_i) \quad (6)$$

By definition of t_o, at most $(m-1)$ tasks are active at time-instant t_o. Consequently, ***there are at most*** $(m-1)$ ***tasks*** τ_i ***that contribute at amount*** $I_2(\tau_i)$***, and the remaining*** $(n - m+1)$ ***tasks must contribute*** $I_1(\tau_i)$. Hence Equation 2 may be rewritten as follows:

$$\sum_{\tau_i \in \tau} I_1(\tau_i) + \sum_{\text{the } (m-1) \text{ largest}} I_{\text{DIFF}}(\tau_i) > m(A_k + D_k - C_k) \quad (7)$$

Observe that all the terms in Equation 7 above are completely defined for a given task system, once a value is chosen for A_k. Hence for a deadline miss of τ_k to occur, there must exist some A_k such that Equation 7 is satisfied. Conversely, in order for all deadlines of τ_k to be met it is sufficient that Equation 7 be violated for all values of A_k. Theorem 2 follows immediately:

Theorem 2 *Task system τ is* EDF*-schedulable upon m unit-capacity processors if for all tasks $\tau_k \in \tau$ and all $A_k \geq 0$,*

$$\left(\sum_{\tau_i \in \tau} I_1(\tau_i) + \sum_{\textit{the } (m-1) \textit{ largest}} I_{\text{DIFF}}(\tau_i) \right) \leq m(A_k + D_k - C_k) \quad (8)$$

where $I_1(\tau_i)$ and $I_{\text{DIFF}}(\tau_i)$ are as defined in Equations 3 and 6 respectively. ■

The earlier tests — the density, [BAK] and [BCL] tests — are not exact even for uniprocessor systems (the special case when $m = 1$). (This is also indicated by the fact that all these prior tests have polynomial run-time, while EDF-schedulability analysis of sporadic task systems on uniprocessors is not known to be in polynomial time.) The following corollary asserts that our test is superior to earlier tests in this regard:

Corollary 1 *The* EDF *schedulability test of Theorem 2 is a generalization of the exact uniprocessor* EDF *schedulability test of [9].*

Proof Sketch: For $m = 1$, there are $(m-1) = 0$ tasks that are active at time-instant t_o; i.e., t_o is the classical "idle instant" of uniprocessor real-time scheduling theory. By adding C_k to both the LHS and the RHS of Condition 8, it can be shown that the LHS reduces to the sum of the demand bound functions of all tasks over an interval of size $A_k + D_k$, and the RHS reduces to the interval length. Hence, when $m = 1$ Condition 8 is asserting that the cumulative processor demand over all intervals must not exceed the interval length, which is exactly what the uniprocessor EDF schedulability test of [9] checks. ■

Corollary 1 does not imply that we have obtained an exciting new result showing that uniprocessor EDF-schedulability analysis can be done in polynomial time; Section 6.1 shows that our algorithm has pseudo-polynomial run-time when system utilization is bounded.

6.1 Run-time complexity

For given τ_k and A_k, it is easy to see that Condition 8 can be evaluated in time linear in n:

- Compute $I_1(\tau_i)$, $I_2(\tau_i)$, and $I_{\text{DIFF}}(\tau_i)$ for each i — total time is $\mathcal{O}(n)$.
- Use linear-time selection [12] on $\{I_{\text{DIFF}}(\tau_1), I_{\text{DIFF}}(\tau_2), \ldots, I_{\text{DIFF}}(\tau_n)\}$ to determine the $(m-1)$ tasks that contribute to the second sum on the LHS.

How many values of A_k must be tested, in order for us to be able to ascertain that Condition 8 is satisfied for all $A_k \geq 0$? Theorem 3 below provides the answer.

Theorem 3 *If Condition 8 is to be violated for any A_k, then it is violated for some A_k satisfying the condition below:*

$$A_k \leq \frac{C_\Sigma - D_k(m - U(\tau)) + \sum_i (T_i - D_i)U_i + mC_k}{m - U(\tau)} \quad (9)$$

where C_Σ denotes the sum of the $(m-1)$ largest C_i's.

Proof: It is easily seen that $I_1(\tau_i) \leq \text{DBF}(\tau_i, A_k + D_k)$, and $I_2(\tau_i) \leq \text{DBF}(\tau_i, A_k + D_k) + C_i$. From this, it can be shown that the LHS of Condition 8 is $\leq C_\Sigma + \sum_{\tau_i \in \tau} \text{DBF}(\tau_i, A_k + D_k)$.

For this to exceed the RHS of Condition 8, it is necessary that

$$C_\Sigma + \text{DBF}(\tau, A_k + D_k) > m(A_k + D_k - C_k)$$

$$\Rightarrow \text{(bounding DBF using the technique of [9])}$$

$$C_\Sigma + (A_k + D_k)U(\tau) + \sum_i (T_i - D_i)U_i > m(A_k + D_k - C_k)$$

$$\equiv C_\Sigma + D_k U(\tau) + \sum_i (T_i - D_i)U_i - m(D_k - C_k) > A_k(m - U(\tau))$$

$$\equiv A_k \leq \frac{C_\Sigma - D_k(m - U(\tau)) + \sum_i (T_i - D_i)U_i + mC_k}{m - U(\tau)}$$

which is as claimed in the theorem. ■

It can also be shown that Condition 8 need only be tested at those values of A_k at which $\text{DBF}(\tau_i, A_k + D_k)$ changes for some τ_i. Corollary 2 follows.

Corollary 2 *The condition in Theorem 2 above can be tested in time pseudo-polynomial in the task parameters, for all task systems τ for which $U(\tau)$ is bounded by a constant strictly less than the number of processors m.* ■

Comparison to the [BAK] and [BCL] tests. By design, our test runs slower than the [BAK], [BCL], and density tests: our test has pseudo-polynomial run-time complexity while all the other tests have polynomial complexity. In practice, we would expect that all tests be used within a single framework: a system be first tested for schedulability using the polynomial-time tests, and only those that are not determined to be schedulable by any of these tests be subjected to our slower test.

It is fairly easy to construct task systems that are determined to be schedulable by only our test (and not the other tests); indeed, Corollary 1 above demonstrates that our test is the only one that is optimal upon uniprocessors (i.e., when $m = 1$). Furthermore, preliminary simulation experiments indicate that our test significantly outperforms the other tests upon task systems in which

- The number of tasks n is significantly greater than the number of processors m (i.e., $n \gg m$). This is probably a consequence of the fact that our test only considers carry-in from $m - 1$ tasks, while all the prior tests must account for carry-in from all n tasks.
- The parameters of the different tasks may be of widely varying orders of magnitude. Once again, this is by design: our test was specifically designed to handle such systems.

7 Conclusions

Recently, much attention has been focused upon the scheduling of systems of recurring tasks on multiprocessor platforms. A fairly deep and detailed understanding has been obtained concerning the multiprocessor scheduling of Liu and Layland task systems (see, e.g., [13] for a survey).

Encouraged by this success, researchers have been considering multiprocessor scheduling of task systems represented using more general models. Here the record is not quite as positive – despite some recent successes, this field of study has not yielded many results. In particular, while some encouraging results have been obtained concerning partitioned scheduling, our knowledge of global scheduling remains rudimentary for recurring real-time task systems represented in models more general than the Liu and Layland model. One of the observations that has come out of all this research is that *global scheduling is fundamentally different from, and seems much more difficult than, partitioned scheduling.*

In this paper, we have described some of our recent findings concerning global multiprocessor scheduling. We have attempted to identify and highlight some of the fundamental issues that render global schedulability analysis so difficult. Building upon the techniques of Baker [2, 3] and of Bertogna et al. [10], we have designed a global-EDF schedulability test that is quite a bit more sophisticated than earlier tests, and that has overcome several of the deficiencies of these earlier tests.

References

[1] AUDSLEY, N. C. *Flexible Scheduling in Hard-Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, 1993.

[2] BAKER, T. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2003), IEEE Computer Society Press, pp. 120–129.

[3] BAKER, T. P. An analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel and Distributed Systems 16*, 8 (2005), 760–768.

[4] BAKER, T. P. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Tech. Rep. TR-050601,

Department of Computer Science, Florida State University, 2005.

[5] BAKER, T. P. An analysis of fixed-priority schedulability on a multiprocessor. *Real-Time Systems: The International Journal of Time-Critical Computing 32*, 1–2 (2006), 49–71.

[6] BAKER, T. P. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Proceeding of the International Conference on Real-Time and Network Systems* (Poitiers, France, 2006).

[7] BARUAH, S., AND FISHER, N. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium* (Miami, Florida, December 2005), IEEE Computer Society Press.

[8] BARUAH, S., AND FISHER, N. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers 55*, 7 (July 2006), 918–923.

[9] BARUAH, S., MOK, A., AND ROSIER, L. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium* (Orlando, Florida, 1990), IEEE Computer Society Press, pp. 182–190.

[10] BERTOGNA, M., CIRINEI, M., AND LIPARI, G. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Palma de Mallorca, Balearic Islands, Spain, July 2005), IEEE Computer Society Press, pp. 209–218.

[11] BERTOGNA, M., CIRINEI, M., AND LIPARI, G. New schedulability tests for real-time tasks sets scheduled by deadline monotonic on multiprocessors. In *Proceedings of the 9th International Conference on Principles of Distributed Systems* (Pisa, Italy, December 2005), IEEE Computer Society Press.

[12] BLUM, M., FLOYD, R. W., PRATT, V., RIVEST, R. L., AND TARJAN, R. E. Time bounds for selection. *Journal of Computer and System Sciences 7*, 4 (Aug. 1973), 448–461.

[13] CARPENTER, J., FUNK, S., HOLMAN, P., SRINIVASAN, A., ANDERSON, J., AND BARUAH, S. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, J. Y.-T. Leung, Ed. CRC Press LLC, 2003.

[14] DERTOUZOS, M. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress* (1974), pp. 807–813.

[15] DHALL, S. *Scheduling Periodic Time-Critical Jobs on Single Processor and Multiprocessor Systems*. PhD thesis, Department of Computer Science, The University of Illinois at Urbana-Champaign, 1977.

[16] FISHER, N., AND BARUAH, S. Global static-priority scheduling of sporadic task systems on multiprocessor platforms. In *Proceeding of the IASTED International Conference on Parallel and Distributed Computing and Systems* (Dallas, TX, November 2006), IASTED.

[17] FISHER, N., BARUAH, S., AND BAKER, T. The partitioned scheduling of sporadic tasks according to static priorities. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Dresden, Germany, July 2006), IEEE Computer Society Press.

[18] GOOSSENS, J., FUNK, S., AND BARUAH, S. Priority-driven scheduling of periodic task systems on multiprocessors. *Real Time Systems 25*, 2–3 (2003), 187–205.

[19] JOSEPH, M., AND PANDYA, P. Finding response times in a real-time system. *The Computer Journal 29*, 5 (Oct. 1986), 390–395.

[20] LEUNG, J., AND MERRILL, M. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters 11* (1980), 115–118.

[21] LEUNG, J., AND WHITEHEAD, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2* (1982), 237–250.

[22] LIU, C., AND LAYLAND, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (1973), 46–61.

[23] MOK, A. K. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[24] RIPOLL, I., CRESPO, A., AND MOK, A. K. Improvement in feasibility testing for real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing 11* (1996), 19–39.