The Partitioned Multiprocessor Scheduling of Deadline-Constrained Sporadic Task Systems

Sanjoy Baruah, *Member*, *IEEE*, and
Nathan Fisher, *Student Member*, *IEEE*

Abstract—A polynomial-time algorithm is presented for partitioning a collection of sporadic tasks, each constrained to have its relative-deadline parameter be no larger than its period parameter, among the processors of an identical multiprocessor platform. Since the partitioning problem is easily seen to be NP-hard in the strong sense, this algorithm is unlikely to be optimal. A quantitative characterization of its worst-case performance is provided in terms of *resource augmentation*: It is shown that any set of sporadic tasks that can be partitioned among the processors of an m-processor identical multiprocessor platform will be partitioned by this algorithm on an m-processor platform in which each processor is $(3 - 1/m)$ times as fast.

Index Terms—Sporadic tasks, partitioned scheduling, multiprocessors, resource augmentation.

1 Introduction

Over the years, the sporadic task model [19] has proven remarkably useful for the modeling of recurring processes that occur in hard-real-time systems. In this model, a *sporadic task* $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* e_i, a *(relative) deadline* d_i, and a *minimum interarrival separation* p_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 p_i time units. Each job has a worst-case execution requirement equal to e_i and a deadline that occurs d_i time units after its arrival time. 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 $\tau_i = (e_i, d_i, p_i)$ 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: $d_i \leq p_i$.

A sporadic task system is said to be *feasible* upon a specified platform if it is possible to schedule the system on the platform 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. The feasibility-analysis of sporadic task systems on preemptive *uni*processors has been extensively studied. It is known (see, e.g., [6]) that a sporadic task system is feasible on a preemptive uniprocessor if and only if all deadlines can be met when each task in the system has a job arrive at the same time-instant and subsequent jobs arrive as rapidly as legal (such a combination of job-arrival sequences is sometimes referred to as a *synchronous arrival sequence* for the sporadic task system). This fact, in conjunction with the optimality of the Earliest Deadline First scheduling algorithm (EDF) for scheduling preemptive uniprocessor systems [16], [8], has allowed for the design of preemptive uniprocessor feasibility-analysis algorithms for sporadic task systems [19], [6].

On **multiprocessor** systems, two alternative paradigms for scheduling collections of sporadic tasks have been considered: *partitioned* and *global* scheduling. In the partitioned approach, the tasks are statically partitioned among the processors, i.e., each task is assigned to a processor and is always executed on it. Under global scheduling, it is permitted that a job that has previously been preempted from one processor resume execution at a later point in time upon a different processor, at no additional cost (however, each job may be executing on at most one processor at each instant in time).

Most prior research on multiprocessor scheduling of collections of sporadic tasks has assumed that *all tasks have their deadlines equal to their period parameters* (i.e., $d_i = p_i$ for all tasks τ_i)—such sporadic systems are sometimes referred to in the literature as **implicit-deadline** systems.[1] For implicit-deadline systems, feasibility-analysis under the global paradigm is trivial [10], [21]: An implicit-deadline system τ is feasible upon a platform comprised of m unit-capacity processors if and only if 1) $(e_i/p_i) \leq 1$ for each task $\tau_i \in \tau$ and 2) $\sum_{\tau_i \in \tau}(e_i/p_i) \leq m$. Under the partitioned paradigm, feasibility-analysis for implicit-deadline systems can be transformed to a bin-packing problem [12] and shown to be NP-hard in the strong sense; sufficient feasibility tests for various bin-packing heuristics have recently been obtained [18], [17], [9].

The research described in this report is part of a larger project that is aimed at obtaining a better understanding of the multiprocessor scheduling of sporadic task systems comprised of tasks that do not satisfy the "$d_i = p_i$" constraint. We are motivated to perform this research for two major reasons. First, sporadic task systems that do not necessarily satisfy the implicit-deadline constraint often arise in practice in the modeling of real-time application systems and it therefore behooves us to have a better understanding of the behavior of such systems. Second, we observe that, in the case of *uni*processor real-time scheduling, moving from implicit-deadline systems (the initial work of Liu and Layland [16]) to constrained systems (as represented in, e.g., [19], [14], [15], [13], [2] etc.,[2] had a major impact on the maturity and development of the field of uniprocessor real-time systems; we are hopeful that progress in better understanding the multiprocessor scheduling of constrained sporadic task systems will result in a similar improvement in our ability to build and analyze multiprocessor real-time application systems.

In this paper, we report our findings concerning the preemptive multiprocessor scheduling of constrained sporadic real-time systems under the partitioned paradigm. Since the process of partitioning tasks among processors reduces a multiprocessor scheduling problem to a series of uniprocessor problems (one to each processor), the optimality of EDF for preemptive uniprocessor scheduling [16], [8] makes EDF a reasonable algorithm to use as the runtime scheduling algorithm on each processor. Therefore, we henceforth make the assumption that each processor and the tasks assigned to it by the partitioning algorithm are scheduled during runtime according to EDF and focus on the partitioning problem.

The constrained task model is a generalization of the implicit-deadline model; therefore, the intractability result (feasibility analysis being NP-hard in the strong sense) continues to hold. To our knowledge, there have been no prior nontrivial positive theoretical results concerning this problem—"trivial" results include the obvious ones that τ is feasible on m processors if 1) it is feasible on a single processor or 2) the system obtained by replacing each task τ_i by a task $\tau_i' = (e_i, d_i, d_i)$ is deemed feasible using the heuristics presented in [18], [17]. Our major contribution is a polynomial-time algorithm for partitioning a given sporadic

• *The authors are with the Department of Computer Science, Campus Box 3175, Sitterson Hall, University of North Carolina, Chapel Hill, NC 27599-3175. E-mail: baruah@cs.unc.edu.*

Manuscript received 9 May 2005; revised 14 Nov. 2005; accepted 31 Jan. 2006; published online 22 May 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0152-0505.

1. A notable and important exception is the recent work of Baker [3], [4], [5], which considers systems of sporadic tasks with $d_i \neq p_i$. We will discuss the relationship between our results and the work of Baker in Section 3.
2. This is merely a small sample and by no means an exhaustive list of citations of important and influential papers concerning the uniprocessor scheduling of constrained sporadic task systems.

task system among a specified number of processors, that makes the following performance guarantee:

If a sporadic task system is feasible on m identical processors, then the same task system can be partitioned by our algorithm among m identical processors *in which the individual processors are* $(3 - \frac{1}{m})$ *times as fast* as in the original system such that all jobs of all tasks assigned to each processor will always meet their deadlines if scheduled using the preemptive EDF scheduling algorithm.

Organization. The remainder of this paper is organized as follows: In Section 2, we review some properties of the *demand bound function*—an abstract characterization of sporadic tasks by the maximum workload such tasks are able to generate over a time interval of given length. In Section 3, we position our findings within a larger context of multiprocessor real-time scheduling theory and compare and contrast our results with related research. In Section 4, we present, and prove correct, our partitioning algorithm. In Section 5, we prove that our algorithm does indeed satisfy the property claimed above: If given sufficiently faster processors, it is able to guarantee to meet all deadlines for all feasible systems.

2 The Demand Bound Function

For any sporadic task τ_i and any real number $t \geq 0$, the *demand bound function* $\mathrm{DBF}(\tau_i, t)$ is the largest cumulative execution requirement of all jobs that can be generated by τ_i to have both their arrival times and their deadlines within a contiguous interval of length t. It has been shown [6] that the cumulative execution requirement of jobs of τ_i over an interval $[t_o, t_o + t)$ is maximized if one job arrives at the start of the interval—i.e., at time-instant t_o—and subsequent jobs arrive as rapidly as permitted—i.e., at instants $t_o + p_i, t_o + 2p_i, t_o + 3p_i, \ldots$. Equation (1) below follows directly [6]:

$$\mathrm{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \max\left(0, \left(\left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) \times e_i\right). \tag{1}$$

An approximation of $\mathrm{DBF}(\tau_i, t)$. The function $\mathrm{DBF}(\tau_i, t)$, if plotted as a function of t for a given task τ_i, is represented by a series of "steps," each of height e_i, at time-instants d_i, $d_i + p_i$, $d_i + 2p_i$, $\cdots$, $d_i + kp_i$, $\cdots$. Albers and Slomka [1] have proposed a technique for *approximating* the DBF, which tracks the DBF exactly through the first several steps and then approximates it by a line of slope e_i/p_i. In the following, we are applying this technique in essentially tracking DBF exactly for a *single* step of height e_i at time-instant d_i, followed by a line of slope e_i/p_i.

$$\mathrm{DBF}^*(\tau_i, t) = \begin{cases} 0, & \text{if } t < d_i \\ e_i + u_i \times (t - d_i), & \text{otherwise.} \end{cases} \tag{2}$$

As stated earlier, it has been shown that the cumulative execution requirement of jobs of τ_i over an interval is maximized if one job arrives at the start of the interval, and subsequent jobs arrive as rapidly as permitted. Intuitively, approximation DBF^* (2) models this job-arrival sequence by requiring that the first job's deadline be met explicitly by being assigned e_i units of execution between its arrival-time and its deadline and that τ_i be assigned $u_i \times \Delta t$ of execution over time-interval $[t, t + \Delta t)$, for all instants t after the deadline of the first job and for arbitrarily small positive Δt.

Observe that the following inequalities hold for all τ_i and for all $t \geq 0$:

$$\mathrm{DBF}(\tau_i, t) \leq \mathrm{DBF}^*(\tau_i, t) < 2 \cdot \mathrm{DBF}(\tau_i, t), \tag{3}$$

with the ratio $\mathrm{DBF}^*(\tau_i, t)/\mathrm{DBF}(\tau_i, t)$ being maximized just prior to the deadline of the second job of τ_i—i.e., at $t = d_i + p_i - \epsilon$ for ϵ an arbitrarily small positive number—in the synchronous arrival sequence; at this time-instant, $\mathrm{DBF}^*(\tau_i, t) \to 2e$ while $\mathrm{DBF}(\tau_i, t) = e$.

3 Context and Related Work

Given the specifications of a multiprocessor sporadic task system, **feasibility analysis** is the process of determining whether it is possible for the system to meet all timing constraints under all legal combinations of job arrivals. While feasibility analysis is a very interesting and important question from a theoretical perspective, the following question is more relevant to the system designer: *Given the specifications of a multiprocessor sporadic task system and a scheduling algorithm that will be used to schedule the system during runtime, how do we determine whether the system will meet all its deadlines when scheduled using this scheduling algorithm?* Note that, at least for the designer of hard-real-time systems, this question must be answered beforehand, during the design of the system and prior to system runtime.

With respect to *global* scheduling, there is a result due to Phillips et al. [20], which can be paraphrased as follows:

If a collection of real-time jobs is feasible on m identical processors, then the same collection of jobs will be scheduled to meet all deadlines by the global EDF scheduling algorithm on m identical processors in which the individual processors are $(2 - \frac{1}{m})$ times as fast as in the original system.

That is, global EDF is a runtime scheduling algorithm that can guarantee to successfully schedule on m identical processors any set of jobs that is feasible on m processors $m/(2m-1)$ times as fast as those available to EDF.[3]

Our results, in this paper, are with respect to *partitioned* scheduling of sporadic task systems. One of the implications of our results is (see Corollary 1):

If a sporadic task system is (global or partitioned) feasible on m identical processors, then the same sporadic task system will be scheduled to always meet all deadlines by the partitioned EDF scheduling algorithm on m identical processors in which the individual processors are $(3 - \frac{1}{m})$ times as fast as in the original system.

If we *know* that a given sporadic task system is feasible on m processors of a given computing capacity, these results indicate that, based on current knowledge, global EDF is to be preferred to partitioned EDF since the processor speedup needed is less by a factor of $(3 - \frac{1}{m})/(2 - \frac{1}{m}) = \lim_{m\to\infty} 1.5$. (Of course, this discussion is overly simplistic in that it ignores all the other factors that have a role to play in the making of this decision, such as the relative cost of inter-processor migrations, etc.)

Despite this additional speedup that is potentially needed, however, it turns out that our partitioned EDF result is more useful to the designer of actual real-time application systems than the global EDF result, for the following reason: There currently is no (exact, or nontrivial sufficient) test known for determining whether a given sporadic task system is (global) feasible on a given number of processors; hence, we have no way of checking the antecedent to either of the results above—the global EDF one of [20], or our partitioned EDF result in Corollary 1—and thereby determining the minimum-speed processors that would suffice for EDF scheduling under either paradigm. In the absence of such a feasibility test, a system designer must verify that a *chosen* runtime scheduling algorithm will always meet all deadlines. This can be done for EDF using the tests of Baker [3], [5] or the one recently proposed by Bertogna et al. [7]—these tests accept as input the specifications of task system τ and determine whether τ will be scheduled to meet all deadlines upon a multiprocessor platform when scheduled using global EDF. However, these tests are sufficient rather than exact and none of them offer a resource augmentation guarantee: For any speedup factor f, there are feasible task systems which these tests will fail to certify as being global-EDF-schedulable upon platforms with processors that are f times as fast as the one upon which the system is feasible. The algorithm that

3. Note that this result is very general in that it applies to any collection of jobs and not just only to jobs generated by a system of sporadic tasks.

```
PARTITION($\tau, m$)
    ▷ The collection of sporadic tasks $\tau = \{\tau_1, \ldots, \tau_n\}$ is to be partitioned on $m$ identical, unit-capacity
      processors denoted $\pi_1, \pi_2, \ldots, \pi_m$. (Tasks are indexed according to non-decreasing value of relative
      deadline parameter: $d_i \leq d_{i+1}$ for all $i$.) $\tau(\pi_k)$ denotes the tasks assigned to processor $\pi_k$; initially,
      $\tau(\pi_k) \leftarrow \varnothing$ for all $k$.
1   for $i \leftarrow 1$ to $n$
    ▷ $i$ ranges over the tasks, which are indexed by non-decreasing value of the deadline parameter
2       for $k \leftarrow 1$ to $m$
        ▷ $k$ ranges over the processors, considered in any order
3           if $\tau_i$ satisfies Condition 4 on processor $\pi_k$ then
                ▷ assign $\tau_i$ to $\pi_k$; proceed to next task
4               $\tau(\pi_k) \leftarrow \tau(\pi_k) \bigcup \{\tau_i\}$
5               goto line 7
6       end (of inner for loop)
7       if $(k > m)$ return PARTITIONING FAILED
8   end (of outer for loop)
9   return PARTITIONING SUCCEEDED
```

Fig. 1. Pseudocode for partitioning algorithm.

we present in Section 4 guarantees that the task mapping produced by it will indeed always meet all deadlines. (An additional issue, somewhat orthogonal to our discussion above, is this: If one is indeed scheduling under the partitioned paradigm, the optimality of uniprocessor EDF justifies its use as the algorithm for scheduling each processor after performing the partitioning. By contrast, multiprocessor global EDF is provably not optimal even for scheduling implicit-deadline periodic task systems; hence there is no good reason known why EDF should be chosen as the runtime scheduling algorithm under global scheduling.)

4 A Partitioning Algorithm

Given sporadic task system τ comprised of n tasks $\tau_1, \tau_2, \ldots \tau_n$ and a platform comprised of m unit-capacity processors $\pi_1, \pi_2, \ldots, \pi_m$, we now describe an algorithm for partitioning the tasks in τ among the m processors. With no loss of generality, let us assume that *tasks are indexed according to nondecreasing order of their relative deadline parameter* (i.e., $d_i \leq d_{i+1}$ for all i, $1 \leq i < n$). Our partitioning algorithm considers the tasks in the order $\tau_1, \tau_2, \ldots$. Suppose that tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ have all been successfully allocated among the m processors and we are now attempting to allocate task τ_i to a processor. Our algorithm for doing this is a variant of the *First Fit* [11] algorithm for bin-packing and is as follows: For any processor π_ℓ, let $\tau(\pi_\ell)$ denote the tasks from among $\tau_1, \ldots, \tau_{i-1}$ that have already been allocated to processor π_ℓ. Considering the processors $\pi_1, \pi_2, \ldots, \pi_m$, in any order, we will assign task τ_i to the first processor π_k, $1 \leq k \leq m$, that satisfies the following two conditions:

$$\left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, d_i)\right) \geq e_i \tag{4}$$

and

$$\left(1 - \sum_{\tau_j \in \tau(\pi_k)} u_j\right) \geq u_i. \tag{5}$$

If no such π_k exists, then we declare failure: We are unable to conclude that sporadic task system τ is feasible upon the m-processor platform. (This algorithm is presented in pseudocode form in Fig. 1.)

The following lemma asserts that, in assigning a task τ_i to a processor π_k, our partitioning algorithm does not adversely affect the feasibility of the tasks assigned earlier to each processor.

Lemma 1. *If the tasks previously assigned to each processor were EDF-feasible on that processor and our algorithm assigns task τ_i to processor π_k, then the tasks assigned to each processor (including processor π_k) remain EDF-feasible on that processor.*

Proof. Observe that the EDF-feasibility of the processors other than processor π_k is not affected by the assignment of task τ_i to processor π_k. It remains to demonstrate that, if the tasks assigned to π_k were EDF-feasible on π_k prior to the assignment of τ_i and Conditions 4 and 5 are satisfied, then the tasks on π_k remain EDF-feasible after adding τ_i.

The scheduling of processor π_k after the assignment of task τ_i to it is a uniprocessor scheduling problem. It is known (see, e.g., [6]) that a uniprocessor system of preemptive sporadic tasks is feasible if and only all deadlines can be met for the synchronous arrival sequence (i.e., when each task has a job arrive at the same time-instant and subsequent jobs arrive as rapidly as legal). Also, recall that EDF is an optimal preemptive uniprocessor scheduling algorithm. Hence, to demonstrate that π_k remains EDF-feasible after adding task τ_i to it, it suffices to demonstrate that all deadlines can be met for the synchronous arrival sequence. Our proof of this fact is by contradiction. That is, we suppose that a deadline is missed at some time-instant t_f, when the synchronous arrival sequence is scheduled by EDF, and derive a contradiction which leads us to conclude that this supposition is incorrect, i.e., no deadline is missed.

Observe that t_f must be $\geq d_i$ since it is assumed that the tasks assigned to π_k are EDF-feasible prior to the addition of τ_i, and τ_i's first deadline in the critical arrival sequence is at time-instant d_i.

By the *processor demand criterion* for preemptive uniprocessor feasibility (see, e.g., [6]), it must be the case that

$$\mathrm{DBF}(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}(\tau_j, t_f) > t_f,$$

from which it follows, since DBF^* is always an upper bound on DBF, that

$$\mathrm{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, t_f) > t_f. \tag{6}$$

Since tasks are considered in order of nondecreasing relative deadline, it must be the case that all tasks $\tau_j \in \tau(\pi_k)$ have $d_j \leq d_i$. We therefore have, for each $\tau_j \in \tau(\pi_k)$,

$$\begin{aligned}\mathrm{DBF}^*(\tau_j, t_f) &= e_j + u_j(t_f - d_j) \quad \text{(By definition)} \\ &= e_j + u_j(d_i - d_j) + u_j(t_f - d_i) \\ &= \mathrm{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i).\end{aligned} \tag{7}$$

Furthermore,

$$\begin{aligned}&\mathrm{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, t_f) \\ &\equiv (e_i + u_i(t_f - d_i)) + \quad (\textit{By definition and } (7) \textit{ above}) \\ &\quad \left(\sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i)\right) \\ &\equiv \left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, d_i)\right) \\ &\quad + (t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right).\end{aligned}$$

Consequently, (6) above can be rewritten as follows:

$$\begin{aligned}&\left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, d_i)\right) + \\ &(t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) > (t_f - d_i) + d_i.\end{aligned} \tag{8}$$

However, by Condition 4, $(e_i + \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, d_i)) \leq d_i$); (8) therefore implies

$$(t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) > (t_f - d_i),$$

which in turn implies that

$$\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) > 1,$$

which contradicts Condition 5. □

Lemma 2 below asserts that it actually suffices to test only Condition 4, rather than having to test both Condition 4 and Condition 5.

Lemma 2. *For constrained sporadic task systems, any τ_i satisfying Condition 4 satisfies Condition 5 as well.*

Proof. For constrained-deadline sporadic task systems, any τ_i satisfying Condition 4 satisfies Condition 5 as well. To see this, observe (from (2)) that, for any constrained task τ_k, for all $t \geq d_k$, it is the case that

$$\mathrm{DBF}^*(\tau_k, t) = u_k \times (t + p_k - d_k) \geq u_k \times t.$$

Hence,

$$\begin{aligned}&\text{Condition 4} \\ &\equiv \left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, d_i) \geq e_i\right) \\ &\Rightarrow d_i - \sum_{\tau_j \in \tau(\pi_k)} (u_j \times d_i) \geq e_i \\ &\Rightarrow 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq \frac{e_i}{d_i} \\ &\Rightarrow 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq u_i \\ &\equiv \text{Condition 5.}\end{aligned}$$

□

The correctness of the partitioning algorithm follows, by repeated applications of Lemma 1:

Theorem 1. *If our partitioning algorithm returns PARTITIONING SUCCEEDED on task system τ, then the resulting partitioning is EDF-feasible.*

Proof Sketch. Observe that the algorithm returns PARTITIONING SUCCEEDED if and only if it has successfully assigned each task in τ to some processor.

Prior to the assignment of task τ_1, each processor is trivially EDF-feasible. It follows from Lemma 1 that all processors remain EDF-feasible after each task assignment assignment as well. Hence, all processors are EDF-feasible once all tasks in τ have been assigned. □

4.1 Runtime Complexity

In attempting to map task τ_i, observe that our partitioning algorithm essentially evaluates, in Condition 4, the workload generated by the previously mapped $(i-1)$ tasks on each of the m processors. Since $\mathrm{DBF}^*(\tau_j, t)$ can be evaluated in constant time (see (2)), a straightforward computation of this workload would require $\mathcal{O}(i+m)$ time. Hence, the runtime of the algorithm in mapping all n tasks is no more than $\sum_{i=1}^{n} \mathcal{O}(i+m)$, which is $\mathcal{O}(n^2)$ under the reasonable assumption that $m \leq n$.

5 Evaluation

As stated in Section 1, our partitioning algorithm represents a sufficient, rather than exact, test for feasibility—it is possible that there are systems that are feasible under the partitioned paradigm but which will be incorrectly flagged as "infeasible" by our partitioning algorithm. Indeed, this is to be expected since a simpler problem—partitioning collections of sporadic tasks that all have their deadline parameters equal to their period parameters—is known to be NP-hard in the strong sense while our algorithm runs in $\mathcal{O}(n^2)$ time. In this section, we offer a quantitative evaluation of the efficacy of our algorithm. Specifically, we derive some properties (Theorem 2 and Corollary 1) of our partitioning algorithm, which characterize its performance. We would like to stress that *these properties are not intended to be used as feasibility tests to determine whether our algorithm would successfully schedule a given sporadic task system*—since our algorithm itself runs efficiently in polynomial time, the "best" (i.e., most accurate) polynomial-time test for determining whether a particular system is successfully scheduled by our algorithm is to actually run the algorithm and check whether it performs a successful partition or not. Rather, these properties are intended to provide a quantitative measure of how effective our partitioning algorithm is vis a vis the performance of an optimal scheduler.

For a given task system $\tau = \{\tau_1, \ldots, \tau_n\}$, let us define the following notation:

$$\delta_{\max} \overset{\text{def}}{=} \max_{i=1}^{n}(e_i/d_i), \tag{9}$$

$$\delta_{\text{sum}} \overset{\text{def}}{=} \max_{t>0}\left(\frac{\sum_{j=1}^{n}\text{DBF}(\tau_j, t)}{t}\right), \tag{10}$$

$$u_{\max} \overset{\text{def}}{=} \max_{i=1}^{n}(u_i), \tag{11}$$

$$u_{\text{sum}} \overset{\text{def}}{=} \sum_{j=1}^{n} u_j. \tag{12}$$

Intuitively, the larger of $\delta_{\max}$ and $u_{\max}$ represents the maximum computational demand of any *individual* task and the larger of δ_{sum} and u_{sum} represents the maximum cumulative computational demand of all the tasks in the system. (For constrained task system τ, observe that $\delta_{\max} \geq u_{\max}$ and $\delta_{\text{sum}} \geq u_{\text{sum}}$.) Lemma 3 follows immediately.

Lemma 3. *If task system τ is feasible (under either the partitioned or the global paradigm) on an identical multiprocessor platform comprised of m processors of computing capacity ξ each, it must be the case that*

$$\xi \geq \delta_{\max}$$

and

$$m \cdot \xi \geq \delta_{\text{sum}}.$$

Proof. Observe that each job of each task of τ can receive at most $\xi \cdot d_i$ units of execution by its deadline; hence, we must have $e_i \leq \xi \cdot d_i$. Taken over all tasks in τ, this observation yields the first condition.

The requirement that $m\xi \geq \delta_{\text{sum}}$ is obtained by considering a sequence of job arrivals for τ that defines δ_{sum}, i.e., a sequence of job arrivals over an interval $[0, t_o)$ such that

$$\frac{\sum_{j=1}^{n}\text{DBF}(\tau_j, t_o)}{t_o} = \delta_{\text{sum}}.$$

The total amount of execution that all these jobs may receive over $[0, t_o)$ is equal to $m \cdot \xi \cdot t_o$; hence, $\delta_{\text{sum}} \leq m \cdot \xi$. □

Lemma 3 above specifies necessary conditions for our partitioning algorithm to successfully partition a sporadic task system; Theorem 2 below specifies a *sufficient* condition. But first, a technical lemma that will be used in the proof of Theorem 2.

Lemma 4. *Suppose that our partitioning algorithm is attempting to schedule task system τ on a platform comprised of unit-capacity processors. If $\delta_{\text{sum}} \leq \frac{1}{2}$, then Condition 4 is always satisfied. Consequently, any τ satisfying $\delta_{\text{sum}} \leq \frac{1}{2}$ is successfully partitioned (on any number of processors ≥ 1.)*

Proof. Observe that $\delta_{\text{sum}} \leq \frac{1}{2}$ implies that $\sum_{\tau_j \in \tau}\text{DBF}(\tau_j, t_o) \leq \frac{t_o}{2}$ for all $t_o \geq 0$. By (2), this in turn implies that $\sum_{\tau_j \in \tau}\text{DBF}^*(\tau_j, t_o) \leq t_o$ for all $t_o \geq 0$; specifically, at $t_o = d_i$ when evaluating Condition 4. But, violating Condition 4 requires that $(\text{DBF}^*(\tau_i, d_i) + \sum_{\tau_j \in \tau(\pi_k)}\text{DBF}^*(\tau_j, d_i))$ exceed d_i. □

Thus, any sporadic task system satisfying $\delta_{\text{sum}} \leq \frac{1}{2}$ is successfully scheduled by our algorithm. We now describe, in Theorem 2, what happens when this condition is not satisfied.

Theorem 2. *Any sporadic task system τ is successfully scheduled by our algorithm on m unit-capacity processors, for any*

$$m \geq \left(\frac{2\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}}\right). \tag{13}$$

Proof. Let us suppose that our partitioning algorithm fails to obtain a partition for τ on m unit-capacity processors. In particular, let us suppose that task τ_i cannot be mapped on to any processor.

Since τ_i fails the test of Condition 4 on each processor in Π, it must be the case that each processor $\pi_\ell \in \Pi$ satisfies

$$\sum_{\tau_j \in \tau(\pi_\ell)}\text{DBF}^*(\tau_j, d_i) > (d_i - e_i).$$

Summing over all m such processors and noting that the tasks $\tau_1, \tau_2, \ldots, \tau_i$ are a subset of the tasks in τ, we obtain

$$\begin{aligned} &\sum_{j=1}^{n}\text{DBF}^*(\tau_j, d_i) > m(d_i - e_i) + e_i \\ \Rightarrow \quad &\text{(By (3))} \\ &2\sum_{j=1}^{n}\text{DBF}(\tau_j, d_i) > m(d_i - e_i) + e_i \\ \Rightarrow &\frac{\sum_{j=1}^{n}\text{DBF}(\tau_j, d_i)}{d_i} > \frac{m}{2}\left(1 - \frac{e_i}{d_i}\right) + \frac{e_i}{2d_i}. \end{aligned} \tag{14}$$

By definition of δ_{sum} (10),

$$\frac{\sum_{j=1}^{n}\text{DBF}(\tau_j, d_i)}{d_i} \leq \delta_{\text{sum}}. \tag{15}$$

Chaining (14) and (15) above, we obtain

$$\begin{aligned} &\frac{m}{2}\left(1 - \frac{e_i}{d_i}\right) + \frac{e_i}{2d_i} < \delta_{\text{sum}} \\ &\Rightarrow m < \frac{2\delta_{\text{sum}} - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}. \end{aligned} \tag{16}$$

In order for our algorithm to successfully schedule τ on m processors, it is sufficient that the negation of the above hold:

$$m \geq \frac{2\delta_{\text{sum}} - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}.$$

Since the right-hand side of the above inequality is maximized when $\frac{e_i}{d_i}$ is as large as possible, we get (13)

$$m \geq \frac{2\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}},$$

as a sufficient condition for τ to be successfully scheduled by our algorithm. □

The technique of **resource augmentation** may be used to quantify the "goodness" (or otherwise) of an algorithm for solving problems for which optimal algorithms are either impossible in practice (e.g., because optimal decisions require knowledge of future events) or computationally intractable. In this technique, the performance of the algorithm being discussed is compared with that of a hypothetical optimal one, under the assumption that the algorithm under discussion has access to *more resources* than the optimal algorithm. Using Theorem 2 above, we now present such a result concerning our partitioning algorithm.

Corollary 1. *Our algorithm makes the following performance guarantee:* If a sporadic task system is feasible on m identical processors, each of a particular computing capacity, then our algorithm will successfully partition this system upon a platform comprised of m processors that are each $(3 - \frac{1}{m})$ times as fast as the original.

Proof. Let us assume that $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a sporadic task system that is feasible on m processors each of computing capacity equal to ξ. We will prove below that τ is guaranteed to be successfully partitioned by our partitioning algorithm on m unit-capacity processors, for all values of $\xi \leq \frac{m}{3m-1}$.

Since τ is feasible on m ξ-speed processors, it follows from Lemma 3 that the tasks in τ satisfy the following properties:

$$\delta_{\max} \leq \xi, \text{ and } \delta_{\text{sum}} \leq m \cdot \xi.$$

By substituting these in (13), we get

$$\begin{aligned} & m \geq \frac{2\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}} \\ \Leftarrow\ & m \geq \frac{2m\xi - \xi}{1 - \xi} \\ \equiv\ & \xi \leq \frac{m}{3m-1}, \end{aligned}$$

which is as claimed in Corollary 1. □

5.1 Discussion

We reiterate that the result in Corollary 1 above is *not* intended to be used as a feasibility test to determine whether our algorithm would successfully schedule a given sporadic task system; rather, this property provides a quantitative measure of how effective our partitioning algorithm is.

Observe that there are two points in our partitioning algorithm during which errors may be introduced. First, we are approximating a solution to a generalization of the bin-packing problem. Second, we are approximating the demand bound function DBF by the function DBF*, thereby introducing an additional approximation factor of two ((3)). While the first of these sources of errors arises even in the consideration of implicit-deadline systems, the second is unique to the generalization in the task model. Indeed, it can be shown that

any implicit-deadline sporadic task system τ that is (global or partitioned) feasible on m identical processors can be partitioned in polynomial time, using our partitioning algorithm, upon m processors that are $(2 - \frac{1}{m})$ times as fast as the original system, when EDF is used to schedule each processor during runtime.

Thus, in terms of resource augmentation, the generalization of the task model costs us a factor that is always less than 2 and (rapidly and) asymptotically approaches 1.5 as $m \to \infty$, for constrained deadlines.

6 Conclusions

Most prior theoretical research concerning the multiprocessor scheduling of sporadic task systems has imposed the additional constraint that all tasks have their deadline parameter equal to their period parameter. In this work, we have removed this constraint and have considered the scheduling of constrained sporadic task systems upon preemptive multiprocessor platforms, under the partitioned paradigm of multiprocessor scheduling. We have designed an algorithm for performing the partitioning of a given collection of sporadic tasks upon a specified number of processors and have proven the correctness of, and evaluated the effectiveness of, this partitioned algorithm. The techniques we have employed are novel and interesting and we hope that they will be of some use in designing superior, and more efficient, algorithms for analyzing multiprocessor real-time systems.

While we have assumed in this paper that our multiprocessor platform is comprised of identical processors, we observe that our results are easily extended to apply to *uniform multiprocessor* platforms—platforms in which different processors have different speeds or computing capacities—under the assumption that each processor has sufficient computing capacity to be able to accommodate each task in isolation. We are currently working on extending the results presented in this paper to uniform multiprocessor platforms in which this assumption may not hold.

Acknowledgments

This research has been supported in part by the US National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825).

References

[1] K. Albers and F. Slomka, "An Event Stream Driven Approximation for the Analysis of Real-Time Systems," *Proc. EuroMicro Conf. Real-Time Systems,* pp. 187-195, July 2004.

[2] N. Audsley, A. Burns, and A. Wellings, "Deadline Monotonic Scheduling Theory and Application," *Control Eng. Practice,* vol. 1, no. 1, pp. 71-78, 1993.

[3] T. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis," *Proc. IEEE Real-Time Systems Symp.,* pp. 120-129, Dec. 2003.

[4] T.P. Baker, "An Analysis of Deadline-Monotonic Schedulability on a Multiprocessor," Technical Report TR-030201, Dept. of Computer Science, Florida State Univ., 2003.

[5] T.P. Baker, "An Analysis of EDF Schedulability on a Multiprocessor," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 8, pp. 760-768, Aug. 2005.

[6] S. Baruah, A. Mok, and L. Rosier, "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor," *Proc. 11th Real-Time Systems Symp.,* pp. 182-190, 1990.

[7] M. Bertogna, M. Cirinei, and G. Lipari, "Improved Schedulability Analysis of EDF on Multiprocessor Platforms," *Proc. EuroMicro Conf. Real-Time Systems,* pp. 209-218, July 2005.

[8] M. Dertouzos, "Control Robotics: The Procedural Control of Physical Processors," *Proc. IFIP Congress,* pp. 807-813, 1974.

[9] S. Funk and S. Baruah, "Task Assignment on Uniform Heterogeneous Multiprocessors," *Proc. EuroMicro Conf. Real-Time Systems,* pp. 219-226, July 2005.

[10] W. Horn, "Some Simple Scheduling Algorithms," *Naval Research Logistics Quarterly,* vol. 21, pp. 177-185, 1974.

[11] D. Johnson, "Fast Algorithms for Bin Packing," *J. Computer and Systems Science,* vol. 8, no. 3, pp. 272-314, 1974.

[12] D.S. Johnson, "Near-Optimal Bin Packing Algorithms," PhD thesis, Dept. of Math., Massachusetts Inst. of Technology, 1973.

[13] J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Tasks with Arbitrary Deadlines," *Proc. IEEE Real-Time Systems Symp.,* pp. 201-209, Dec. 1990.

[14] J. Leung and M. Merrill, "A Note on the Preemptive Scheduling of Periodic, Real-Time Tasks," *Information Processing Letters,* vol. 11, pp. 115-118, 1980.

[15] J. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation,* vol. 2, pp. 237-250, 1982.

[16] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, 1973.

[17] J.M. Lopez, J.L. Diaz, and D.F. Garcia, "Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems," *Real-Time Systems: The Int'l J. Time-Critical Computing,* vol. 28, no. 1, pp. 39-68, 2004.

[18] J.M. Lopez, M. Garcia, J.L. Diaz, and D.F. Garcia, "Worst-Case Utilization Bound for EDF Scheduling in Real-Time Multiprocessor Systems," *Proc. EuroMicro Conf. Real-Time Systems,* pp. 25-34, June 2000.

[19] A.K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," PhD thesis, Laboratory for Computer Science, Massachusetts Inst. of Technology, 1983, Technical Report No. MIT/LCS/TR-297.

[20] C.A. Phillips, C. Stein, E. Torng, and J. Wein, "Optimal Time-Critical Scheduling via Resource Augmentation," *Proc. 29th Ann. ACM Symp. Theory of Computing,* pp. 140-149, May 1997.

[21] A. Srinivasan, "Efficient and Flexible Fair Scheduling of Real-Time Tasks on Multiprocessors," PhD thesis, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, 2003.