The Partitioned Dynamic-priority Scheduling of Sporadic Task Systems

Abstract

A polynomial-time algorithm is presented for partitioning a collection of sporadic tasks among the processors of an identical multiprocessor platform. Since the partitioning problem is 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*.

Keywords: Sporadic tasks; partitioned scheduling; multiprocessors.

1 Introduction

Over the years, the sporadic task model (Mok, 1983) has proven remarkably useful for the modelling 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 inter-arrival 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, $1 \leq i \leq n$.

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. (Baruah et al., 1990)) 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 (Liu and Layland, 1973; Dertouzos, 1974), has allowed for the design of preemptive uniprocessor feasibility-analysis algorithms for sporadic task systems (Mok, 1983; Baruah et al., 1990).

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].

[1]Some recent work (Baker, 2003, 2005a; Bertogna et al., 2005a,b), considers systems of sporadic tasks with $d_i \neq p_i$. We will discuss the relationship between our results and this work in Section 3.

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 *arbitrary* sporadic task systems – i.e., 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 modelling 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 (Liu and Layland, 1973)) to arbitrary systems (as represented in, e.g. (Mok, 1983; Leung and Merrill, 1980; Leung and Whitehead, 1982; Lehoczky, 1990; Audsley et al., 1993)), had a major impact in the maturity and development of the field of uniprocessor real-time systems; we are hopeful that progress in better understanding the multiprocessor scheduling of arbitrary 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 arbitrary 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 (Liu and Layland, 1973; Dertouzos, 1974) makes EDF a reasonable algorithm to use as the run-time 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.

Summary of Contributions. We propose two polynomial-time partitioning algorithms. Both partitioning algorithms are variants of the *First-Fit* bin-packing heuristic (Johnson, 1973). The first algorithm we propose assigns sporadic tasks to processors based on their *density* parameter (density is defined as the execution requirement of a task divided by the minimum of either the period or relative deadline parameter). The second partitioning algorithm we propose uses an approximation to the demand-bound function (introduced in Section 2) for a sporadic task and task utilization (utilization is defined as execution requirement divided by the period parameter) as dual criteria for assigning a task to a processor.

For the partitioning algorithm using the demand-bound function approximation, we derive sufficient conditions for success of our algorithm (Theorems 4 and 5). We also show (Corollary 2) that our demand-based partitioning has 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* $(4 - \frac{2}{m})$ *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.

For the density-based partitioning algorithm, we also derive sufficient conditions for success (Theorem 1). However, we show (Theorem 2) that density-based partitioning does not have a performance guarantee on the necessary speed of each processor for the partitioning algorithm to succeed (i.e. there cannot exist a guarantee for density-based partitioning similar to the preceding guarantee for demand-based partitioning).

Organization. The remainder of this paper is organized as follows. In Section 2, we formally specify the task model used in the remainder of this paper, and 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, prove the correctness of, and evaluate the performance of a very simple density-based partitioning algorithm. In Section 5, we present, and prove correct, a somewhat more sophisticated demand-based partitioning algorithm. In Section 6, we prove that this algorithm satisfies a property that the simpler algorithm does not possess: if given sufficiently faster processors, it is able to guarantee to meet all deadlines for all feasible systems. We also list some other results that we have obtained, and include a brief discussion of the significance of our results. In Section 7 we propose an improvement to the algorithm: although this improvement does not seem to effect the worst-case behavior of the algorithm, it is shown to be of use in successfully partitioning some sporadic task systems that our basic algorithm – the one presented in Section 5 – fails to handle.

2 Task and machine 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 inter-arrival separation* p_i. The ratio $u_i \stackrel{\text{def}}{=} e_i/p_i$ of sporadic task τ_i is often referred to as the *utilization* of τ_i, and the ratio $\lambda_i \stackrel{\text{def}}{=} e_i/\min(p_i, d_i)$, as the *density* of τ_i.

We will assume that we have a multiprocessor platform comprised of m identical processors π_1, π_2, …, π_m, on which we are to schedule a system τ of n sporadic tasks: $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$, with $\tau_i = (e_i, d_i, p_i)$ for all i, $1 \le i \le n$. Depending upon what additional restrictions are placed on the relationship between the values of d_i and p_i for each sporadic task $\tau_i \in \tau$, we may define special subclasses of sporadic task systems:

- Sporadic task systems in which each task satisfies the additional constraint that $d_i = p_i$ for each task τ_i are referred to as **implicit-deadline** sporadic tasks systems.

- Sporadic task systems in which each task satisfies the additional constraint that $d_i \leq p_i$ for each task τ_i are often referred to as **constrained** sporadic tasks systems.

Where necessary, the term **arbitrary** sporadic task system will be used to refer to task systems which do not satisfy the above constraints.

The demand bound function

For any sporadic task τ_i and any real number $t \geq 0$, the *demand bound function* $\text{DBF}(\tau_i, t)$ 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 (Baruah et al., 1990) 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 (Baruah et al., 1990):

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

An approximation of $\text{DBF}(\tau_i, t)$. The function $\text{DBF}(\tau_i, t)$, 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 (Albers and Slomka, 2004) 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 (see Figure 1). 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.

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

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^* (Equation 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 $\delta\, t$ (see Figure 2 for a pictorial depiction).

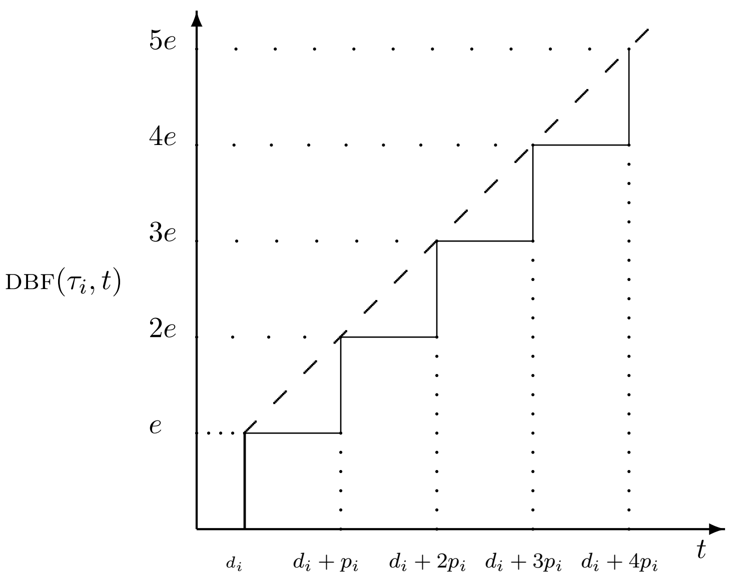

Figure 1: **The step function denotes a plot of** DBF(τ_i, t) **as a function of** t**. The dashed line represents the function** DBF$^*(\tau_i, t)$**, approximating** DBF(τ_i, t)**; for** $t < d_i$**,** DBF$^*(\tau_i, t) \equiv$ DBF$(\tau_i, t) = 0$**.**

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

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

with the ratio DBF$^*(\tau_i, t)/$DBF(τ_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, DBF$^*(\tau_i, t) \rightarrow 2e$ while DBF$(\tau_i, t) = e$.

Comparison of DBF* **and the density approximation.** It is known that a sufficient condition for a sporadic task system to be feasible upon a unit-capacity *uni*processor is $(\sum_{\tau_i \in \tau} \lambda_i) \leq 1$; i.e., the sum of the densities of all tasks in the system not exceed one (see, e.g., (Liu, 2000, Theorem 6.2)). This condition is obtained by essentially approximating the demand bound function DBF(τ_i, t) of τ_i by the quantity $(t \cdot \lambda_i)$. This approximation is never superior to our approximation DBF*, and is inferior if $d_i \neq p_i$: for our example in Figure 1, this approximation would be represented by a straight line with slope e_i/d_i passing through the origin.

3 Context and Related Work

As stated in the introduction, our research objective is to obtain a better understanding of the multiprocessor scheduling of sporadic task systems. Currently not too much is known about this topic; other than the research reported in (Baker, 2003, 2005a; Bertogna et al., 2005a,b), we are

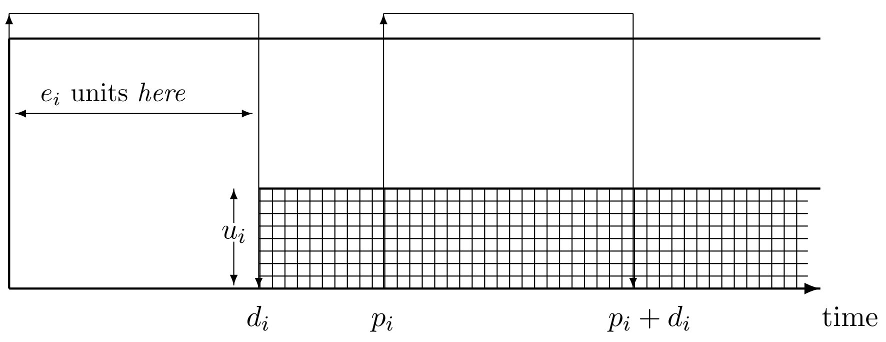

Figure 2: **Pictorial representation of task τ_i's reservation of computing capacity in a processor-sharing schedule. e_i units of execution are reserved over the interval $[0, d_i)$. The hatched region depicts τ_i's reservation of computing capacity over the interval $[d_i, \infty)$ – over any time interval $[t, t+\delta\, t)$, an amount $\delta\, t \times u_i$ of computing capacity is reserved.**

unaware of any theoretical research that specifically addresses multiprocessor scheduling of sporadic tasks in any model other than the implicit-deadline task model. Below, we summarize the results obtained in this paper, and place it within the larger context of the multiprocessor scheduling of sporadic task systems. It is convenient to consider such context from two perspectives: *feasibility/ schedulability analysis*, and *resource augmentation*.

Feasibility and schedulability analysis

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.

For **implicit-deadline** systems, partitioned feasibility-analysis can be transformed to a bin-packing problem (Johnson, 1973) and shown to be NP-hard in the strong sense; sufficient feasibility tests for various bin-packing heuristics have recently been obtained (Lopez et al., 2000, 2004). The heuristics developed in (Lopez et al., 2000, 2004) are guaranteed to determine that a implicit-deadline task system is partitioned-feasible if the total system utilization does not exceed $\frac{\beta m+1}{\beta+1}$ where $\beta \overset{\text{def}}{=} \left\lfloor \frac{1}{u_{\max}(\tau)} \right\rfloor$ and $u_{\max}(\tau) \overset{\text{def}}{=} \max_{\tau_i \in \tau}(u_i)$. The **constrained** and **arbitrary** task models are generalizations of the implicit-deadline model; therefore, the intractability result continues to hold. However, the bin-packing heuristics and related analysis of (Lopez et al., 2000, 2004) does not hold for the constrained-deadline or arbitrary task models. Furthermore, it is known that utilization is a poor metric for the feasibility of non-implicit-deadline systems (e.g. see (Fisher et al., 2006)). To our knowledge, there have been no prior non-trivial positive theoretical results concerning partitioned feasibility analysis of constrained and arbitrary sporadic task systems — "trivial" results include the obvious ones that τ is feasible on m processors if **(i)** it is feasible on a single processor; or **(ii)** the system obtained by replacing each task τ_i by a task $\tau_i' = (e_i, \min(d_i, p_i), \min(d_i, p_i))$ is deemed feasible using the heuristics presented in (Lopez et al., 2000, 2004).

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 run-time, how do we determine whether the system will meet all its deadlines when scheduled using this scheduling algorithm*[2]*?* More formally, for any scheduling algorithm A, a real-time system is said to be ***A*-schedulable** if the system meets all its deadlines during run-time when scheduled using algorithm A. (Note that, at least for the designer of hard real-time systems, schedulability analysis must be performed beforehand during the design of the system, and prior to run-time.)

In the work of (Baker, 2003, 2005a; Bertogna et al., 2005a,b), schedulability analysis techniques have been derived for global scheduling algorithms such as EDF and Deadline-Monotonic (DM). In this paper, our focus is instead on the partitioned scheduling paradigm; that is, we focus on the partitioning of non-implicit-deadline sporadic task systems. In Sections 4 and 5, we present two different algorithms for partitioning sporadic task systems upon multiprocessor platforms. These algorithms both guarantee that all jobs of all tasks assigned to each processor will always meet all deadlines if each processor is scheduled during run-time using the preemptive uniprocessor EDF scheduling algorithm. Hence, both these partitioning algorithms comprise efficient constructive schedulability tests for systems of sporadic tasks — as discussed below, they differ from each other in the kinds of performance guarantee that they are able to make.

Resource augmentation

If a scheduling algorithm is not able to successfully schedule *every* feasible system, how do we evaluate how good it is? One possible approach is to perform extensive simulations – this is the approach adopted in, for example, (Bertogna et al., 2005a; Baker, 2005b, 2006) in evaluating global multiprocessor EDF. Baker (Baker, 2006) has performed empirical analysis of various partitioning algorithms (including the partitioning algorithm we propose in this paper), and has shown that our algorithm performs favorably in comparison to EDF-based global and partitioned scheduling approaches. As our proposed partitioning algorithm has been previously evaluated by Baker (Baker, 2006), we use an alternative analysis approach to quantitatively bound the *degree* to which the algorithm under consideration may underperform a hypothetical optimal one. To obtain such a

[2]Such analysis is sometimes referred to as **schedulability analysis** – in contrast to feasibility analysis, such schedulability analysis is specifically concerned with system behavior under specific scheduling algorithms.

bound, we use the technique of **resource augmentation** to relate the concepts of feasibility and A-schedulability, for specific algorithms A. The technique is as follows: given that a system is known to be feasible upon a particular platform, can we guarantee that algorithm A will always successfully schedule the system if we *augment* the platform in some particular, quantifiable, way (e.g., by increasing the speed, or the number, of processors available to A as compared to the platform on which the system was shown to be feasible)?

With regard to the global paradigm of multiprocessor scheduling, there is a resource-augmentation result due to Phillips et al. (Phillips et al., 1997), which, in the context of sporadic task systems, can be paraphrased as follows:

> **R1:** If a system of sporadic tasks is feasible under the global paradigm on m identical processors, then this system of sporadic tasks is global EDF-schedulable on m identical processors in which the individual processors are $(2 - \frac{1}{m})$ times as fast as in the original system.

That is, if we were able to show that a given sporadic task system were global feasible on a particular platform, then global EDF guarantees to successfully schedule this system on a platform comprised of processors that are approximately twice as fast.

Our results, in this paper, are with respect to the *partitioned* scheduling of sporadic task systems. The algorithm presented in Section 4 is shown to be incapable of making any kind of resource-augmentation performance guarantee. By contrast, the partitioning algorithm presented in Section 5 does offer a resource-augmentation performance guarantee: in Corollary 2 (Section 6) we derive the following result :

> **R2:** If a system of sporadic tasks is feasible under the partitioned paradigm (and consequently, under the global paradigm as well) on m identical processors, then this system of sporadic tasks is partitioned by the algorithm presented in Section 5 on m identical processors in which the individual processors are $(4-\frac{2}{m})$ times as fast as in the original system, such that each partition is uniprocessor EDF-schedulable. (If the task system is a ***constrained*** sporadic task system, then our algorithm successfully partitions it on m processors that are $(3 - \frac{1}{m})$ times as fast.)

Let us now examine the consequences of results $R1$ and $R2$ above, concerning global and partitioned EDF-schedulability, to the schedulability analysis of systems of sporadic tasks. Since $(4 - 2/m) > (2 - 1/m)$, global EDF offers a superior performance guarantee than partitioned EDF *if a sporadic task system were known to be feasible* (at least from the perspective of the minimum processor speed-up needed to ensure that all timing constraints are met – 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.). However, there currently

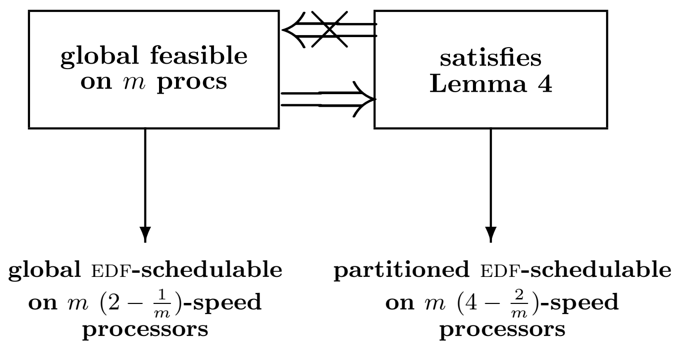
global feasible on m procs
satisfies Lemma 4
global EDF-schedulable on m $(2-\frac{1}{m})$-speed processors
partitioned EDF-schedulable on m $(4-\frac{2}{m})$-speed processors

Figure 3: **How the pieces fit together.**

is no test known for determining whether a given sporadic task system is (global) feasible on a given number of processors; consequently, we do not have an effective procedure for testing the antecedent of result $R1$ for a given task system, and consequently condition $R1$ does not translate to an effective procedure for global EDF-schedulability.

Despite the antecedent of result $R2$ being the same as the antecedent of result $R1$, this turns out to not be an issue for partitioned EDF scheduling. This is because, as we will see, explicitly evaluating the antecedent of $R2$ is not needed in order to be able to apply the partitioning algorithm. The situation is illustrated in Figure 3. We will show in this document that *any* sporadic task system that is global feasible satisfies a certain property (identified in Lemma 4, Section 6), while the converse is not true: satisfying this property is not a sufficient test for global feasibility. While the result $R1$ requires as its antecedent that the sporadic task system actually be feasible under the global paradigm, it turns out that result $R2$ merely requires that the task system satisfy the property identified in Lemma 4 (which is efficiently checkable). Consequently, it is not necessary to have an effective procedure for checking global feasibility in order to take advantage of result $R2$ and obtain an effective procedure for partitioned EDF-scheduling; however, we are not aware of how one could obtain a similar global EDF-schedulability test based on result $R1$.

By composing the result from (Phillips et al., 1997) and the ones we derive in this paper, we can obtain an effective procedure for EDF-schedulability under the global paradigm as follows.

1. Determine, using the algorithms presented in this paper, whether the system is partitioned EDF-schedulable on processors each of speed $m/(2m-1)$ times as fast as the ones available. Any system that is partitioned EDF-schedulable is, by definition, *feasible* under the global paradigm (this is a trivial consequence of the fact that any partitioned schedule is also a global schedule).

2. By result $R1$, this system is consequently global EDF-schedulable on processors that are $(2-\frac{1}{m})$ times as fast as those considered in step 1 above. Since $(2-\frac{1}{m}) \times (m/(2m-1) = 1$, this implies that the system is global EDF-schedulable on the available processors.

Putting these pieces together, we have the following performance guarantee:

> **R3:** If a system of sporadic tasks is feasible under the global paradigm on m identical processors, then this system of sporadic tasks is EDF-schedulable on m identical processors in which the individual processors are $(2 - \frac{1}{m}) \times (4 - \frac{2}{m}) = (8 - \frac{6}{m} + \frac{2}{m^2})$ times as fast as in the original system. (If the task system is a ***constrained*** sporadic task system, then a processor speedup of $(2 - \frac{1}{m}) \times (3 - \frac{1}{m}) = (6 - \frac{5}{m} + \frac{1}{m^2})$ suffices.)

Currently, there is no known resource-augmentation performance guarantee associated with the tests of (Baker, 2003, 2005a; Bertogna et al., 2005a,b).

4 Density-based partitioning

In this section, we present a simple, efficient, algorithm for partitioning a sporadic task system among the processors of a multiprocessor platform. We also show that this partitioning algorithm does not offer a resource-augmentation performance guarantee.

Let us suppose that we are 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 $\pi_1, \pi_2, \ldots, \pi_m$. For each task τ_i, recall that its **density** λ_i is defined as follows:

$$\lambda_i \stackrel{\text{def}}{=} \frac{e_i}{\min(d_i, p_i)} \ .$$

We may also define the maximum task system density:

$$\lambda_{\max}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} \Big(\lambda_i\Big) \ .$$

Without loss of generality, let us assume that the tasks in τ are indexed according to non-increasing density: $\lambda_i \geq \lambda_{i+i}$ for all i, $1 \leq i < n$. Our partitioning algorithm considers the tasks in the order $\tau_1, \tau_2, \ldots$. Suppose that tasks τ_1, τ_2, $\ldots$, τ_{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* (Johnson, 1974) algorithm for bin-packing, and is as follows (see Figure 4 for a pseudo-code representation). 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 any processor π_k, $1 \leq k \leq m$, that satisfies the following condition:

$$\lambda_i + \left(\sum_{\tau_j \in \tau(\pi_k)} \lambda_j \right) \leq 1 \tag{4}$$

```
densityPartition($\tau, m$)
    ▷ 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-increasing density: $\lambda_i \geq \lambda_{i+1}$
      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
2           for $k \leftarrow 1$ to $m$
            ▷ $k$ ranges over the processors, considered in any order
3                   if $\tau_i$ satisfies Conditions 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                           break;
6           end (of inner for loop)
7           if $(k > m)$ return PARTITIONING FAILED
8   end (of outer for loop)
9   return PARTITIONING SUCCEEDED
```

Figure 4: Pseudo-code for density-based partitioning algorithm.

Lemma 1 *Algorithm* densityPartition *successfully partitions any sporadic task system τ on $m \geq 1$ processors, that satisfies the following condition:*

$$\sum_{\tau_i \in \tau} \lambda_i \leq m - (m-1) \times \lambda_{\max}(\tau) \tag{5}$$

Proof: Let us suppose that Algorithm densityPartition fails to partition task system τ; in particular, let us assume that it fails to assign task τ_i to any processor, for some $i \leq n$. It must be the case that each of the m processors fails the test of Equation 4; i.e., tasks previously assigned to each processor have their densities summing to more than $(1 - \lambda_i)$. Summing over all m processors, this implies that

$$\begin{aligned} & \sum_{j=1}^{i-1} \lambda_j > m \times (1 - \lambda_i) \\ \equiv \quad & \sum_{j=1}^{i} \lambda_j > m - (m-1)\lambda_i \\ \Rightarrow \quad & \sum_{\tau_i \in \tau} \lambda_i > m - (m-1) \times \lambda_{\max}(\tau) \end{aligned}$$

Hence, any system which Algorithm densityPartition fails to partition must have $\sum_{\tau_i \in \tau} \lambda_i > m - (m-1) \times \max_{\tau_i \in \tau}(\lambda_i)$. The lemma follows. ■

Theorem 1 *Algorithm* DENSITYPARTITION *successfully partitions any sporadic task system τ on $m \geq 2$ processors, that satisfies the following condition:*

$$\sum_{i=1}^{n} \lambda_i \leq \begin{cases} m - (m-1)\lambda_{\max}(\tau) & \text{if } \lambda_{\max}(\tau) \leq \frac{1}{2} \\ \frac{m}{2} + \lambda_{\max}(\tau) & \text{if } \lambda_{\max}(\tau) \geq \frac{1}{2} \end{cases} \tag{6}$$

Proof: Let us define a task τ_i to be **dense** if $\lambda_i > 1/2$. We consider two cases separately.

Case 1: There are no dense tasks. In this case $\lambda_{\max}(\tau) \leq 1/2$, and Equation 6 reduces to

$$\sum_{i=1}^{n} \lambda_i \leq m - (m-1) \times \lambda_{\max}(\tau) \ ;$$

by Lemma 1, this condition is sufficient to guarantee that Algorithm DENSITYPARTITION successfully partitions the tasks in τ.

Case 2: There are dense tasks. In this case $\lambda_{\max}(\tau) > 1/2$, and Equation 6 reduces to

$$\sum_{i=1}^{n} \lambda_i \leq \frac{m}{2} + \lambda_{\max}(\tau) \ . \tag{7}$$

Observe first that any task system satisfying Condition 7 has at most m dense tasks – this follows from the observation that one task will have density equal to $\lambda_{\max}(\tau)$, and at most $(m-1)$ tasks may each have density $> \frac{1}{2}$ while observing the $\sum_{i=1}^{n} \lambda_i$ bound of Condition 7. We consider separately the two cases when there are strictly less than m dense tasks, and when there are exactly m dense tasks.

Case 2.1: Fewer than m dense tasks. Let n_h denote the number of dense tasks. Algorithm DENSITYPARTITION assigns each of these n_h tasks to a different processor. We will apply Lemma 1 to prove that Algorithm DENSITYPARTITION successfully assigns the tasks $\{\tau_{n_h+1}, \tau_{n_h+2}, \ldots, \tau_n\}$ on $(m - n_h)$ processors; the correctness of the theorem would then follow from the observation that Algorithm DENSITYPARTITION does in fact have $(m-n_h)$ processors left over after tasks τ_1 through τ_{n_h} have been assigned.

Let us now compute upper bounds on the cumulative and maximum densities of the task system $\{\tau_{n_h+1}, \tau_{n_h+2}, \ldots, \tau_n\}$:

$$\sum_{j=n_h+1}^{n} \lambda_j = \sum_{j=1}^{n} \lambda_j - \sum_{j=1}^{n_h} \lambda_j$$

$$\Rightarrow \quad \sum_{j=n_h+1}^{n} \lambda_j \leq \frac{m}{2} + \lambda_{\max}(\tau) - \sum_{j=1}^{n_h} \lambda_j$$

$$\Rightarrow \quad \text{(from the fact that } \lambda_1 = \lambda_{\max}(\tau) \text{ and } \lambda \geq \frac{1}{2} \text{ for all } i = 2, \ldots, n_h\text{)}$$

$$\begin{aligned}
& \sum_{j=n_h+1}^{n} \lambda_j \leq \frac{m}{2} + \lambda_{\max}(\tau) - (\lambda_{\max}(\tau) + \frac{1}{2} \times (n_h - 1)) \\
\equiv\ & \sum_{j=n_h+1}^{n} \lambda_j \leq \frac{m - n_h + 1}{2} \qquad (8)
\end{aligned}$$

Furthermore, since each task in $\{\tau_{n_h+1}, \tau_{n_h+2}, \ldots, \tau_n\}$ is, by definition of n_h, not dense, it is the case that

$$\max_{j=n_h+1}^{n} \left(\lambda_j\right) \leq \frac{1}{2} \qquad (9)$$

By applying the bounds derived in Equation 8 and Equation 9 above to Lemma 1, we may conclude that Algorithm densityPartition successfully assigns the tasks $\{\tau_{n_h+1}, \tau_{n_h+2}, \ldots, \tau_n\}$ on $(m-n_h)$ processors:

$$\begin{aligned}
& \sum_{j=n_h+1}^{n} \lambda_j \leq (m - n_h) - (m - n_h - 1) \max_{j=n_h+1}^{n} \left(\lambda_j\right) \\
\Leftarrow\ & \frac{m - n_h + 1}{2} \leq (m - n_h) - \frac{m - n_h - 1}{2} \\
\equiv\ & \frac{m - n_h + 1}{2} \leq \frac{m - n_h + 1}{2}
\end{aligned}$$

which is, of course, always true.

Case 2.2: Exactly m dense tasks. Let λ_R denote the cumulative densities of all the non-dense tasks: $\lambda_R \stackrel{\text{def}}{=} \sum_{j=m+1}^{n} \lambda_j$. Observe that $\lambda_R < \frac{m}{2} - (m-1) \times \frac{1}{2} = \frac{1}{2}$. By Equation 6, since $\lambda_{\max}(\tau) > \frac{1}{2}$, the total system density does not exceed $\frac{m}{2} + \lambda_{\max}(\tau)$. Because $\lambda_1 = \lambda_{\max}(\tau)$, the $(m-1)$ tasks $\tau_2, \tau_3, \ldots, \tau_m$ have total density $\leq \frac{m}{2} - \lambda_R$, it must be the case that λ_m is at most $(\frac{m}{2} - \lambda_R)/(m-1)$. We will prove that $\lambda_m + \lambda_R \leq 1$, from which it will follow that all the tasks $\tau_m, \ldots, \tau_n$ fit on a single processor. Indeed,

$$\begin{aligned}
& \lambda_m + \lambda_R \leq 1 \\
\Leftarrow\ & \frac{\frac{m}{2} - \lambda_R}{m-1} + \lambda_R \leq 1 \\
\equiv\ & m - 2\lambda_R + 2m\lambda_R - 2\lambda_R \leq 2m - 2 \\
\equiv\ & \lambda_R(2m - 4) \leq m - 2 \\
\equiv\ & \lambda_R \leq \frac{1}{2}
\end{aligned}$$

which is true.

■

4.1 Resource augmentation analysis

Consider the task system τ comprised of the n tasks $\tau_1, \tau_2, \ldots, \tau_n$, with τ_i having the following parameters:

$$e_i = 2^{i-1}, \quad d_i = 2^i - 1, \quad p_i = \infty \ .$$

Observe that τ is feasible on a single unit-capacity processor, since

$$\sum_{i=1}^{n} \text{DBF}(\tau_i, t) = \sum_{i=1}^{\lfloor \log_2(t+1) \rfloor} 2^{i-1}$$

which is $\leq t$ for all $t \geq 0$, with equality at $t = 2-1, 2^2-1, \ldots, 2^n - 1$ and strict inequality for all other t.

Now, $\lambda_i > \frac{1}{2}$ for each i; i.e., each task is dense. Hence, Algorithm densityPartition will assign each task to a distinct processor, and hence is only able to schedule τ upon n unit-capacity processors. Alternatively, Algorithm densityPartition would need a processor of computing capacity $\geq n$ in order to have Condition 4 be satisfied for all tasks on a single processor. By making n arbitrarily large, we have the following result:

Theorem 2 *For any constant $\xi \geq 1$, there is a sporadic task system τ and some positive integer m such that τ is (global or partitioned) feasible on m unit-capacity processors, but Algorithm* densityPartition *fails to successfully partition the tasks in τ among* **(i)** *$\xi \times m$ unit-capacity processors, or* **(ii)** *m processors each of computing capacity ξ.* ■

5 Algorithm partition

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 another algorithm for partitioning the tasks in τ among the m processors. In Section 6 we will prove that this algorithm, unlike the one described above in Section 4, does make non-trivial resource-augmentation performance guarantees. With no loss of generality, let us assume that the tasks in τ are indexed according to non-decreasing order of their relative deadline parameter (i.e., $d_i \leq d_{i+1}$ for all i, $1 \leq i < n$). Our partitioning algorithm (see Figure 5 for a pseudo-code representation) considers the tasks in the order $\tau_1, \tau_2, \ldots$. Suppose that tasks τ_1, τ_2, $\ldots$, τ_{i-1} have all been successfully allocated among the m processors, and we are now attempting to allocate task τ_i to a processor. 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, $1 \leq k \leq m$,

```
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: $d_i \le d_{i+1}$ for all $i$.) $\tau(\pi_k)$ denotes the tasks assigned to processor $\pi_k$; initially,
      $\tau(\pi_k) \leftarrow \emptyset$ 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 Conditions 10 and 11 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                           break;
6           end (of inner for loop)
7           if $(k > m)$ return PARTITIONING FAILED
8   end (of outer for loop)
9   return PARTITIONING SUCCEEDED
```

Figure 5: Pseudo-code for partitioning algorithm.

that satisfies the following two conditions:

$$\left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) \ge e_i \tag{10}$$

and

$$\left(1 - \sum_{\tau_j \in \tau(\pi_k)} u_j\right) \ge u_i \ ; \tag{11}$$

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.

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 2 *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 10 and 11 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. (Baruah et al., 1990)) 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., (Baruah et al., 1990)), it must be the case that

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

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

$$\text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) > t_f \ . \tag{12}$$

Since tasks are considered in order of non-decreasing 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 $\tau_j \in \tau(\pi_k)$,

$$\begin{aligned}
\text{DBF}^*(\tau_j, t_f) &= e_j + u_j(t_f - d_j) \qquad \text{(By definition)} \\
&= e_j + u_j(d_i - d_j) + u_j(t_f - d_i) \\
&= \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i) \qquad (13)
\end{aligned}$$

Furthermore,

$$\begin{aligned}
&\text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) \\
\equiv\ &(e_i + u_i(t_f - d_i)) + \qquad \textit{(By Equation 13 above)} \\
&\left(\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i) \right)
\end{aligned}$$

$$\begin{aligned}
\equiv \quad & \left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) \\
& +(t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right)
\end{aligned}$$

Consequently, Inequality 12 above can be rewritten as follows:

$$\begin{aligned}
& \left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) + \\
& \quad (t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) > (t_f - d_i) + d_i
\end{aligned} \tag{14}$$

However by Condition 10, $(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)) \leq d_i)$; Inequality 14 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

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

which contradicts Condition 11. ■

In the special case where the given sporadic task system is known to be **constrained** — i.e., each task's relative deadline parameter is no larger than its period parameter — Lemma 3 below asserts that it actually suffices to test only Condition 10, rather than having to test both Condition 10 and Condition 11.

Lemma 3 *For* **constrained** *sporadic task systems, any τ_i satisfying Condition 10 during the execution of Line 3 in the algorithm of Figure 5 satisfies Condition 11 as well.*

Proof: To see this, observe (from Equation 2) that for any constrained task τ_k, for all $t \geq d_k$ it is the case that

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

Hence,

$$\begin{aligned}
& \text{Condition 10} \\
\equiv \quad & \left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \geq e_i\right) \\
\Rightarrow \quad & d_i - \sum_{\tau_j \in \tau(\pi_k)} (u_j \times d_i) \geq e_i
\end{aligned}$$

$$\begin{aligned}\Rightarrow \quad & 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq \frac{e_i}{d_i} \\ \Rightarrow \quad & 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq u_i \\ & \equiv \text{Condition } 11\end{aligned}$$

■

Hence if the task system τ being partitioned is known to be a constrained task system, we need only check Condition 10 (rather than both Condition 10 and Condition 11) on line 3 in Figure 5. The correctness of the partitioning algorithm follows, by repeated applications of Lemma 2:

Theorem 3 *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 2 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. ■

Run-time complexity

In attempting to map task τ_i, observe that our partitioning algorithm essentially evaluates, in Equations 10 and 11, the workload generated by the previously-mapped $(i-1)$ tasks on each of the m processors. Since $\text{DBF}^*(\tau_j, t)$ can be evaluated in constant time (see Equation 2), a straightforward computation of this workload would require $\mathcal{O}(i+m)$ time. Hence, the run-time 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 $m \leq n$.

6 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 5 and Corollary 2) 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 given task system $\tau = \{\tau_1, \ldots, \tau_n\}$, let us define the following notation:

$$\delta_{\max}(\tau) \overset{\text{def}}{=} \max_{i=1}^{n} (e_i/d_i) \tag{15}$$

$$\delta_{\text{sum}}(\tau) \overset{\text{def}}{=} \max_{t>0} \left(\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, t)}{t} \right) \tag{16}$$

$$u_{\max}(\tau) \overset{\text{def}}{=} \max_{i=1}^{n} (u_i) \tag{17}$$

$$u_{\text{sum}}(\tau) \overset{\text{def}}{=} \sum_{j=1}^{n} u_j \tag{18}$$

(For constrained-deadline sporadic task systems, observe that $\delta_{\max}(\tau)$ as defined above is just the largest density of any task in τ; however, $\delta_{\text{sum}}(\tau)$ is not the sum of the densities of all the tasks in τ. Consider, for instance, the constrained-deadline sporadic task system introduced in Section 4.1: Consider the task system τ comprised of the n tasks $\tau_1, \tau_2, \ldots, \tau_n$, with τ_i having the following parameters:

$$e_i = 2^{i-1}, \quad d_i = 2^i - 1, \quad p_i = \infty \ .$$

While the sum of the densities for this system is equal to

$$1 + \frac{2}{3} + \frac{4}{7} + \ldots \frac{2^{n-1}}{2^n - 1}$$

and consequently approaches ∞ as $n \to \infty$, $\delta_{\text{sum}}(\tau)$ is equal to 1 for all values of n.)

Intuitively, the larger of $\delta_{\max}(\tau)$ and $u_{\max}(\tau)$ represents the maximum computational demand of any *individual* task, and the larger of $\delta_{\text{sum}}(\tau)$ and $u_{\text{sum}}(\tau)$ represents the maximum cumulative computational demand of all the tasks in the system.

Lemma 4 follows immediately.

Lemma 4 *If task system τ is feasible (under either the partitioned or the global paradigm) on an identical multiprocessor platform comprised of m_o processors of computing capacity ξ each, it must* *be the case that*

$$\xi \geq \max(\delta_{\max}(\tau), u_{\max}(\tau)) \ ,$$

and

$$m_o \cdot \xi \geq \max(\delta_{\text{sum}}(\tau), u_{\text{sum}}(\tau)) \ .$$

Proof: Observe that

1. 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 $e_i \leq \xi \cdot d_i$.
2. No individual task's utilization may exceed the computing capacity of a processor; i.e., it must be the case that $u_i \leq \xi$.

Taken over all tasks in τ, these observations together yield the first condition.

In the second condition, the requirement that $m_o \xi \geq u_{\text{sum}}(\tau)$ simply reflects the requirement that the cumulative utilization of all the tasks in τ not exceed the computing capacity of the platform. The requirement that $m_o \xi \geq \delta_{\text{sum}}(\tau)$ is obtained by considering a sequence of job arrivals for τ that defines $\delta_{\text{sum}}(\tau)$; i.e., a sequence of job arrivals over an interval $[0, t_o)$ such that $\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, t_o)}{t_o} = \delta_{\text{sum}}(\tau)$. The total amount of execution that all these jobs may receive over $[0, t_o)$ is equal to $m_o \cdot \xi \cdot t_o$; hence, $\delta_{\text{sum}}(\tau) \leq m_o \cdot \xi$. ■

Lemma 4 above specifies necessary conditions for our partitioning algorithm to successfully partition a sporadic task system; Theorem 5 below specifies a *sufficient* condition. But first, a technical lemma that will be used in the proof of Theorem 5.

Lemma 5 *Suppose that our partitioning algorithm is attempting to schedule task system τ on a platform comprised of unit-capacity processors.*

C1: *If $u_{\text{sum}}(\tau) \leq 1$, then Condition 11 is always satisfied.*

C2: *If $\delta_{\text{sum}}(\tau) \leq \frac{1}{2}$, then Condition 10 is always satisfied.*

Proof: The proof of C1 is straightforward, since violating Condition 11 requires that $(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j)$ exceed 1.

To see why C2 holds as well, observe that $\delta_{\text{sum}}(\tau) \leq \frac{1}{2}$ 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 Inequality 3, 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 10. But, violating Condition 10 requires that $(\text{DBF}^*(\tau_i, d_i) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i))$ exceed d_i. ■

Corollary 1

1. *Any sporadic task system τ satisfying $(u_{\text{sum}}(\tau) \leq 1 \bigwedge \delta_{\text{sum}}(\tau) \leq \frac{1}{2})$ is successfully partitioned on any number of processors ≥ 1.*

2. *Any constrained sporadic task system τ satisfying $(\delta_{\text{sum}}(\tau) \leq \frac{1}{2})$ is successfully partitioned on any number of processors ≥ 1.*

Proof Sketch: Part 1 follows directly from Lemma 5, since Line 3 of the partitioning algorithm in Figure 5 will always evaluate to "true."

Part 2 follows from Lemmas 5 and 3. By Lemma 3, we need only determine that Condition 10 is satisfied, in order to ensure that Line 3 of the partitioning algorithm in Figure 5 evaluate to "true." By Condition C2 of Lemma 5, this is ensured by having $\delta_{\text{sum}}(\tau) \leq \frac{1}{2}$. ■

Thus, any sporadic task system satisfying both $u_{\text{sum}}(\tau) \leq 1$ and $\delta_{\text{sum}}(\tau) \leq \frac{1}{2}$ is successfully scheduled by our algorithm. We now describe, in Theorem 5, what happens when one or both these conditions are not satisfied.

Lemma 6 *Let m_1 denote the number of processors, $0 \leq m_1 \leq m$, on which Condition 10 fails when the partitioning algorithm is attempting to map task τ_i. It must be the case that*

$$m_1 < \frac{2\delta_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} \tag{19}$$

Proof: Since τ_i fails the test of Condition 10 on each of the m_1 processors, it must be the case that each such processor π_ℓ satisfies

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

Summing over all m_1 such processors and noting that the tasks on these processors is a subset of the tasks in τ, we obtain

$$\begin{aligned}
& \sum_{j=1}^{n} \text{DBF}^*(\tau_j, d_i) > m_1(d_i - e_i) + e_i \\
\Rightarrow \quad & \text{(By Inequality 3)} \\
& 2\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) > m_1(d_i - e_i) + e_i \\
\Rightarrow \quad & \frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} > \frac{m_1}{2}(1 - \frac{e_i}{d_i}) + \frac{e_i}{2d_i}
\end{aligned} \tag{20}$$

By definition of $\delta_{\text{sum}}(\tau)$ (Equation 16)

$$\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} \leq \delta_{\text{sum}}(\tau) \tag{21}$$

Chaining Inequalities 20 and 21 above, we obtain

$$\frac{m_1}{2}(1 - \frac{e_i}{d_i}) + \frac{e_i}{2d_i} < \delta_{\text{sum}}(\tau)$$
$$\Rightarrow \qquad m_1 < \frac{2\delta_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}$$

which is as claimed by the lemma. ■

Lemma 7 *Let m_2 denote the number of processors, $0 \leq m_2 \leq m - m_1$, on which Condition 11 fails (but Condition 10 is satisfied) when the partitioning algorithm is attempting to map task τ_i. It must be the case that*

$$m_2 < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \qquad (22)$$

Proof: Since none of the m_2 processors satisfies Condition 11 for task τ_i, it must be the case that there is not enough remaining utilization on each such processor to accommodate the utilization of task τ_i. Therefore, strictly more than $(1 - u_i)$ of the capacity of each such processor has already been consumed; summing over all m_2 processors and noting that the tasks already assigned to these processors is a subset of the tasks in τ, we obtain the following upper bound on the value of m_2:

$$(1 - u_i)m_2 + u_i < \sum_{j=1}^{n} u_j$$
$$\Rightarrow \qquad m_2 < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i}$$

which is as asserted by the lemma. ■

Theorem 4 *Any <u>constrained</u> sporadic task system τ is successfully scheduled by our algorithm on m unit-capacity processors, for any*

$$m \geq \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \delta_{\max}(\tau)} \qquad (23)$$

Proof: Our proof is by contradiction – we will assume that our algorithm fails to partition task system τ on m processors, and prove that, in order for this to be possible, m must violate Inequality 23 above. Accordingly, 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. By Lemma 3, it must be the case that Condition 10 fails for task τ_i on each of the m processors; i.e., m_1 in the statement of Lemma 6 is equal to the total number of processors m. Consequently, Inequality 19 of Lemma 6 yields

$$m < \frac{2\delta_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} \; .$$

By Corollary 1, it is necessary that $\delta_{\rm sum}(\tau) > \frac{1}{2}$ hold. Since $\delta_{\rm max}(\tau) \leq 1$ (if not, the system is trivially non-feasible), the right-hand side of the above inequality is maximized when $\frac{e_i}{d_i}$ is as large as possible, this implies that

$$m < \frac{2\delta_{\rm sum}(\tau) - \delta_{\rm max}(\tau)}{1 - \delta_{\rm max}(\tau)} \ ,$$

which contradicts Inequality 23 above. ■

Theorem 5 *Any sporadic task system τ is successfully scheduled by our algorithm on m unit-capacity processors, for any*

$$m \geq \left(\frac{2\delta_{\rm sum}(\tau) - \delta_{\rm max}(\tau)}{1 - \delta_{\rm max}(\tau)} + \frac{u_{\rm sum}(\tau) - u_{\rm max}(\tau)}{1 - u_{\rm max}(\tau)} \right) \tag{24}$$

Proof: Once again, our proof is by contradiction – we will assume that our algorithm fails to partition task system τ on m processors, and prove that, in order for this to be possible, m must violate Inequality 24 above.

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. There are two cases we consider: **(i)** Condition 10 fails, and **(ii)** Condition 10 is satisfied (implying that Condition 11 must have failed). Let Π_1 denote the m_1 processors upon which this mapping fails because Condition 10 is not satisfied (hence for the remaining $m_2 \stackrel{\rm def}{=} (m - m_1)$ processors, denoted Π_2, Condition 10 is satisfied but Condition 11 is not).

By Lemma 5 above, m_1 will equal 0 if $\delta_{\rm sum}(\tau) \leq \frac{1}{2}$, while m_2 will equal 0 if $u_{\rm sum}(\tau) \leq 1$. Since we are assuming that the partitioning fails, it is not possible that both $\delta_{\rm sum}(\tau) \leq \frac{1}{2}$ and $u_{\rm sum}(\tau) \leq 1$ hold.

Let us extend previous notation as follows: for any collection of processors Π_x, let $\tau(\Pi_x)$ denote the tasks from among $\tau_1, \ldots, \tau_{i-1}$ that have already been allocated to some processor in the collection Π_x. Lemmas 6 and 7 provide, for task systems on which our partitioning algorithm fails, upper bounds on the values of m_1 (i.e., the number of processors in Π_1) and m_2 (the number of processors in Π_2) in terms of the parameters of the task system τ.

We now consider three separate cases.

Case (i): ($\delta_{\rm sum}(\tau) > \frac{1}{2}$ and $u_{\rm sum}(\tau) \leq 1$). As stated in Lemma 5 (C1), Condition 11 is never violated in this case, and m_2 is consequently equal to zero. From this and Lemma 6 (Inequality 19), we therefore have

$$m < \frac{2\delta_{\rm sum}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} \ .$$

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

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

Since the right-hand side of the above inequality is maximized when $\frac{e_i}{d_i}$ is as large as possible, this implies that

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

which certainly holds for any m satisfying the statement of the theorem (Inequality 24).

Case (ii): ($\delta_{\text{sum}}(\tau) \leq \frac{1}{2}$ and $u_{\text{sum}}(\tau) > 1$). As stated in Lemma 5 (C2), Condition 10 is never violated in this case, and m_1 is consequently equal to zero. From this and Lemma 7 (Inequality 22), we therefore have

$$m < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \ .$$

We once again observe that it is sufficient that the negation of the above hold in order for our algorithm to successfully schedule τ on m processors:

$$m \geq \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \ .$$

Since the right-hand side of the above inequality is maximized when u_i is as large as possible, this implies that

$$m \geq \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - u_{\max}(\tau)} \ ,$$

which once again holds for any m satisfying the statement of the theorem (Inequality 24).

Case (iii): ($u_{\text{sum}}(\tau) > 1$ and $\delta_{\text{sum}}(\tau) > \frac{1}{2}$). In this case, both m_1 and m_2 may be non-zero. From $m_1 + m_2 = m$ and Inequality 22, we may conclude that

$$m_1 > m - \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \tag{25}$$

For Inequalities 25 and 19 to both be satisfied, we must have

$$\frac{2\delta_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} > m - \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i}$$

$$\Rightarrow \quad m < \frac{2\delta_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} + \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \tag{26}$$

Hence for our algorithm to successfully schedule τ, it is sufficient that the negation of Inequality 26 hold:

$$m \geq \left(\frac{2\delta_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} + \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \right) \tag{27}$$

Observe that the right hand-side of Inequality 27 is maximized when u_i and $\frac{e_i}{d_i}$ are both as large as possible; by Inequalities 15 and 16, these are defined to be $u_{\max}(\tau)$ and $\delta_{\max}(\tau)$ respectively. We hence get Inequality 24:

$$m \geq \left(\frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \delta_{\max}(\tau)} + \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - u_{\max}(\tau)} \right)$$

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

Recall that in the technique of resource augmentation, 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 5 above, we now present such a result concerning our partitioning algorithm.

Theorem 6 *Our algorithm makes the following performance guarantees:*

1. *if a constrained sporadic task system is feasible on m_o 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 $(2\frac{m_o}{m} + 1 - \frac{1}{m})$ times as fast as the original.*

2. *if an arbitrary sporadic task system is feasible on m_o 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 $(3\frac{m_o}{m} + 1 - \frac{2}{m})$ times as fast as the original.*

Proof: Let us assume that $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is feasible on m_o processors each of computing capacity equal to ξ. Below, we consider separately the cases when τ is a constrained sporadic task system and an arbitrary sporadic task system:

1. τ is a constrained sporadic task system. Since τ is feasible on m_o ξ-speed processors, it follows from Lemma 4 that the tasks in τ satisfy the following properties:

$$\delta_{\max}(\tau) \leq \xi, \quad \text{and} \quad \delta_{\text{sum}}(\tau) \leq m_o \cdot \xi$$

Suppose that τ is successfully scheduled by our algorithm on m unit-capacity processors. By substituting the inequalities above in Equation 23 of Theorem 4, we get

$$\begin{aligned} & m \geq \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \delta_{\max}(\tau)} \\ \Leftarrow \quad & m \geq \frac{2m_o\xi - \xi}{1 - \xi} \\ \equiv \quad & \xi \leq \frac{m}{2m_o + m - 1} \\ \equiv \quad & \frac{1}{\xi} \geq 2\frac{m_o}{m} + 1 - \frac{1}{m} \end{aligned}$$

which is as claimed in the statement of the theorem.

2. τ is an arbitrary sporadic task system. Since τ is feasible on m_o ξ-speed processors, it follows from Lemma 4 that the tasks in τ satisfy the following properties:

$$\delta_{\max}(\tau) \le \xi, \quad u_{\max}(\tau) \le \xi, \quad \delta_{\text{sum}}(\tau) \le m_o \cdot \xi, \quad \text{and} \quad u_{\text{sum}}(\tau) \le m_o \cdot \xi$$

Suppose once again that τ is successfully scheduled by our algorithm on m unit-capacity processors. By substituting the inequalities above in Equation 24 of Theorem 5, we get

$$\begin{aligned} & m \ge \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \delta_{\max}(\tau)} + \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - u_{\max}(\tau)} \\ \Leftarrow \quad & m \ge \frac{2m_o\xi - \xi}{1-\xi} + \frac{m_o\xi - \xi}{1-\xi} \\ \equiv \quad & \xi \le \frac{m}{3m_o + m - 2} \\ \equiv \quad & \frac{1}{\xi} \ge 3\frac{m_o}{m} + 1 - \frac{2}{m} \end{aligned}$$

which is as claimed in the statement of the theorem. ■

By setting $m_o \leftarrow m$ in the statement of Theorem 6 above, we immediately have the following corollary.

Corollary 2 *Our algorithm makes the following performance guarantees:*

1. *if a constrained 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.*

2. *if an arbitrary 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 $(4 - \frac{2}{m})$ times as fast as the original.*

■

7 A pragmatic improvement

We have made several approximations in deriving the results above. One of these has been the use of the approximation $\text{DBF}^*(\tau_i, t)$ of Equation 2 in Condition 10, to determine whether (the first job of) task τ_i can be accommodated on a processor π_k. We could reduce the amount of inaccuracy introduced here, by refining the approximation: rather than approximating $\text{DBF}(\tau_i, t)$ by a single step followed by a line of slope u_i, we could explicitly have included the first k steps, followed by a line of slope u_i (as proposed by Albers and Slomka (Albers and Slomka, 2004)). For the case $k = 2$, such an approximation, denoted $\text{DBF}'(\tau_i, t)$ here, is as follows:

$$\text{DBF}'(\tau_i, t) = \begin{cases} 0, & \text{if } t < d_i \\ e_i, & \text{if } d_i \leq t < d_i + p_i \\ 2e_i + u_i \times (t - (d_i + p_i)), & \text{otherwise} \end{cases} \quad (28)$$

If we were to indeed use an approximation comprised of k steps, instead of the single-step approximation DBF^*, in Condition 10 in determining whether a processor can accommodate an additional task, we would need to explicitly re-check that the first k deadlines of all tasks previously assigned to the processor continue to be met. This is because it is no longer guaranteed that the new deadlines (those of τ_i) will occur *after* the deadlines of previously-assigned tasks, and hence it is possible that adding τ_i to the processor will result in some previously-added task missing one of its deadlines. However, the benefit of using better approximations is a greater likelihood of determining a system feasible; we illustrate by an example.

Example 1 Suppose that task $\tau_j = (1, 1, 10)$ has already been assigned to processor π_k when task $\tau_i = (1, 2, 20)$ is being considered. Evaluating Condition 10, we have

$$\begin{aligned} & d_i - \text{DBF}^*(\tau_j, 2) \geq e_i \\ \equiv \quad & 2 - 0.1 \times (2 + 10 - 1) \geq 1 \\ \equiv \quad & 2 - 1.1 \geq 1 \end{aligned}$$

which is false; hence, we determine that τ_i fails the test of Condition 10 and cannot be assigned to processor π_k.

However, suppose that we were to instead approximate the demand bound function to two steps rather than one, by suing the function $\text{DBF}'(\ ,\ )$ (Equation 28 above). We would need to consider two deadlines for both the new task τ_i as well as the previously-assigned task τ_j. The deadlines for τ_i are at time-instants 2 and 22, and for τ_j at time-instants 1 and 11. The demand-bound computations at all four deadlines are shown below:

- At $t = 1$: $\text{DBF}'(\tau_j, 1) + \text{DBF}'(\tau_i, 1) = 1 + 0 = 1$, which is ≤ 1.
- At $t = 2$: $\text{DBF}'(\tau_j, 2) + \text{DBF}'(\tau_i, 2) = 1 + 1 = 2$, which is ≤ 2.
- At $t = 11$: $\text{DBF}'(\tau_j, 11) + \text{DBF}'(\tau_i, 11) = 1 + 2 = 3$, which is ≤ 11.
- At $t = 22$: $\text{DBF}'(\tau_j, 22) + \text{DBF}'(\tau_i, 22) = 2 + 3.1 = 5.1$, which is ≤ 22.

Furthermore, τ_i also passes the test of Condition 11, since $u_j + u_i = 0.1 + 0.05$ which is ≤ 1. ■

As this example illustrates, the benefit of using a finer approximation is enhanced feasibility: the feasibility test is less likely to incorrectly declare a feasible system to be infeasible. The cost of this improved performance is run-time complexity: rather than just check Condition 10 at d_i for each processor during the assignment of task τ_i, we must check a similar condition on a total of $(i \times k)$ deadlines over all m processors (observe that this remains polynomial-time, for constant k). Hence in practice, we recommend that the largest value of k that results in an acceptable run-time complexity for the algorithm be used.

From a theoretical perspective, we were unable to obtain a significantly better bound than the ones in Theorem 5 and Corollary 2 by using a finer approximation in this manner.

Discussion

We reiterate that the results in Corollary 2 are *not* intended to be used as feasibility tests to determine whether our algorithm would successfully schedule a given sporadic task system; rather, these properties provide 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 (Inequality 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 run-time.*

Thus, the generalization of the task model costs us a factor of 2 in terms of resource augmentation for arbitrary deadlines, and a factor of less than 2, asymptotically approaching 1.5 as $m \to \infty$, for constrained deadlines.

The purpose of the above discussion on the partitioning of implicit-deadline systems is only intended to identify the sources of the error in the approximation factors for constrained-deadline and arbitrary systems (Corollary 2). In practice, a system designer would use the tighter analysis of (Lopez et al., 2000, 2004) for the partitioning implicit-deadline sporadic tasks systems.

8 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 arbitrary 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 proved 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.

Acknowledgements

We are grateful to the anonymous reviewers of both the conference and journal versions of this paper. Their thoughtful comments and suggestions have helped to improve the quality of this paper.

References

K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 187–195, Catania, Sicily, July 2004. IEEE Computer Society Press.

N. Audsley, A. Burns, and A. Wellings. Deadline monotonic scheduling theory and application. *Control Engineering Practice*, 1(1):71–78, 1993.

T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 120–129. IEEE Computer Society Press, December 2003.

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

T. P. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Technical Report TR-050601, Department of Computer Science, Florida State University, 2005b.

T. P. Baker. 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.

S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 209–218, Palma de Mallorca, Balearic Islands, Spain, July 2005a. IEEE Computer Society Press.

M. Bertogna, M. Cirinei, and G. Lipari. 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 2005b. IEEE Computer Society Press.

M. Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

N. Fisher, S. Baruah, and T. Baker. 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.

D. Johnson. Fast algorithms for bin packing. *Journal of Computer and Systems Science*, 8(3):272–314, 1974.

D. S. Johnson. *Near-optimal Bin Packing Algorithms.* PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1973.

J. P. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–209, Dec. 1990.

J. Leung and M. Merrill. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11:115–118, 1980.

J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

J. W. S. Liu. *Real-Time Systems.* Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 2000.

J. M. Lopez, J. L. Diaz, and D. F. Garcia. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, 28(1):39–68, 2004.

J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-case utilization bound for EDF scheduling in real-time multiprocessor systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 25–34, Stockholm, Sweden, June 2000. IEEE Computer Society Press.

A. K. Mok. *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.

C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, 4–6 May 1997.