The Partitioned Multiprocessor Scheduling of Sporadic Task Systems*

Sanjoy Baruah Nathan Fisher
The University of North Carolina at Chapel Hill

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*: 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* $(4 - 2/m)$ *times as fast.*

1 Introduction

In the **sporadic task model** [12] a 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 *uni*processors has been extensively studied. It is known (see, e.g. [4]) 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 [9, 6], has allowed for the design of preemptive uniprocessor feasibility-analysis algorithms for sporadic task systems [12, 4].

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.

In this paper, we report our findings concerning the preemptive multiprocessor scheduling of sporadic task systems under the partitioned paradigm. Most prior research on this subject (e.g., [11, 10]) 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. Feasibility-analysis for implicit-deadline systems under the partitioned paradigm can be transformed to a bin-packing problem [7] and shown to be NP-hard in the strong sense; sufficient feasibility tests for various bin-packing heuristics have been obtained [11, 10]. Since the task model we are considering in this paper is a generalization of the implicit-deadline model, this intractability result continues to hold.

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 [9, 6] makes EDF a reasonable algorithm to use as the run-time scheduling algorithm on each

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

processor. Therefore, we henceforth make the assumption that each processor is scheduled during runtime according to EDF, and focus on the partitioning problem. One of our major contributions is a polynomial-time algorithm for partitioning a given sporadic 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* $(4 - \frac{2}{m})$ *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.

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 present, prove the correctness of, and evaluate the performance of a very simple partitioning algorithm. In Section 4, we present, and prove correct, a somewhat more sophisticated partitioning algorithm. In Section 5, we prove that this algorithm satisfies the property claimed above, 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. In Section 6, we position our findings within a larger context of multiprocessor real-time scheduling theory, and compare and contrast our results with related research.

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 $\lambda_i \stackrel{\text{def}}{=} e_1/\min(d_i, p_i)$, the *density* of τ_i.

We will assume in this paper 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, $1 \leq i \leq n$. In general, no constraints are placed on the relation between the values of d_i and p_i for any sporadic task τ_i. 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.

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 [4] 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 $t_o + p_i$, $t_o + 2p_i$, $t_o + 3p_i$, ... Equation (1) below follows directly [4]:

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

Albers and Slomka [1] have proposed a technique for *approximating* the DBF; the following approximation to DBF is obtained by applying their technique:

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

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 above) 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 $t \geq 0$:

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

3 Density-based partitioning

As stated in Section 1 above, bin-packing heuristics have previously been applied to the partitioned scheduling of implicit-deadline sporadic task systems. The observation linking bin-packing to partitioned scheduling is as follows: an implicit-deadline sporadic task system is uniprocessor EDF-feasible on a unit-capacity processor if and only if the utilizations of all tasks in the system sum to at most one. Hence, each processor can be modelled as a bin of capacity one, and each implicit-deadline task τ_i as an item of size u_i.

One could extend this analogy between bin-packing and task partitioning to arbitrary sporadic task systems, by making use of the following well-known fact:

Fact 1 *A sporadic task system is uniprocessor* EDF*-feasible on a unit-capacity processor if the* densities *of all tasks in the system sum to at most one.*■

Hence, each processor can be modelled as a bin of capacity one, and each task τ_i as an item of size λ_i. Applying standard bin-packing analysis techniques, one obtains the following result:

Theorem 1 *Let* $\lambda_{\text{sum}} \stackrel{def}{=} \sum_{\tau_i \in \tau} \lambda_i$, *and* $\lambda_{\max} \stackrel{def}{=} \max_{\tau_i \in \tau}\{\lambda_i\}$. *By modelling sporadic tasks as items of size equal to their densities and processors as unit-capacity bins, the First-Fit Decreasing (FFD) bin-packing heuristic [7] successfully partitions any sporadic task system* τ *on* $m \geq 2$ *processors, provided that* τ *and* m *satisfy the following condition:*

$$\lambda_{\text{sum}} \leq \begin{cases} m - (m-1)\lambda_{\max}, & \text{if } \lambda_{\max} \leq \frac{1}{2} \\ \frac{m}{2} + \lambda_{\max} & \text{if } \lambda_{\max} \geq \frac{1}{2} \end{cases} \qquad (4)$$

■

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* (e.g., more processors, or ones of greater computing capacity) than the optimal algorithm.

It turns out that the FFD heuristic performs arbitrarily poorly from a resource-augmentation perspective; this is formally demonstrated in the following theorem:

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 the FFD heuristic 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.

Proof: Let $m \equiv 1$. Consider the task system τ comprised of the $n_o = \lceil \xi \rceil$ tasks $\tau_1, \tau_2, \ldots, \tau_{n_o}$, 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

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

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

Now, $\lambda_i > \frac{1}{2}$ for each i. Therefore, the FFD heuristic assigns each task to a distinct processor, and hence is only able to schedule τ upon n_o unit-capacity processors. Alternatively, FFD would need a processor of computing capacity $\geq n_o$ in order to have all tasks "fit" on a single processor. ■

4 Algorithm PARTITION

In this section, we present a polynomial-time partitioning algorithm that does not suffer from the kind of poor performance identified in Theorem 2. In particular, this algorithm guarantees that it is able to always meet all deadlines for all feasible systems, if given sufficiently faster processors than are needed by a hypothetical optimal algorithm.

With no loss of generality, let us assume in this section 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, $1 \leq i < n$). Algorithm PARTITION considers the tasks in the order $\tau_1, \tau_2, \ldots$. Suppose that tasks τ_1, τ_2, ..., 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. Algorithm PARTITION is a variant of the First Fit [7] algorithm for bin-packing, and is as follows (see Figure 1 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, Algorithm PARTITION assigns task τ_i to the first processor π_k, $1 \leq k \leq m$, that satisfies the following two conditions:

$$\left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \right) \geq e_i \qquad (5)$$

and

$$\left(1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \right) \geq u_i \ ; \qquad (6)$$

If no such π_k exists, then Algorithm PARTITION declares failure: it is 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, Algorithm PARTITION does not adversely affect the feasibility of the tasks assigned earlier to each processor.

```
PARTITION(τ, m)
   ▷ The collection of sporadic tasks τ = {τ_1, ..., τ_n} is to be partitioned on m identical, unit-capacity processors
     denoted π_1, π_2, ..., π_m. (Tasks are indexed according to non-decreasing value of relative deadline parameter:
     d_i ≤ d_{i+1} for all i.) τ(π_k) denotes the tasks assigned to processor π_k; initially, τ(π_k) ← ∅ for all k.
1  for i ← 1 to n
   ▷ i ranges over the tasks, which are indexed by non-decreasing value of the deadline parameter
2          for k ← 1 to m
           ▷ k ranges over the processors, considered in any order
3                  if τ_i satisfies Conditions 5 and 6 on processor π_k then
                           ▷ assign τ_i to π_k; proceed to next task
4                          τ(π_k) ← τ(π_k) ⋃ {τ_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
```

Figure 1. Pseudo-code for partitioning algorithm.

Lemma 1 *If the tasks previously assigned to each processor were* EDF*-feasible on that processor and Algorithm* PARTITION *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 5 and 6 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. [4]) 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., [4]), 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

$$\text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) > t_f \ . \qquad (7)$$

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 $\tau_j \in \tau(\pi_k)$,

$$\begin{aligned} \text{DBF}^*(\tau_j, t_f) &= e_j + u_j(t_f - d_j) \qquad \text{(By definition)} \\ &= e_j + u_j(d_i - d_j) + u_j(t_f - d_i) \\ &= \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i) \qquad (8) \end{aligned}$$

Furthermore,

$$\begin{aligned} &\text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) \\ &\quad \equiv \ (e_i + u_i(t_f - d_i)) + \qquad \textit{(By Equation 8 above)} \\ &\qquad \left(\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i) \right) \\ &\quad \equiv \ \left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \right) \end{aligned}$$

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

Consequently, Inequality 7 above can be rewritten as follows:

$$\left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) + (t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) > (t_f - d_i) + d_i \quad (9)$$

However by Condition 5, $(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)) \leq d_i)$; Inequality 9 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 6. ■

The correctness of Algorithm PARTITION follows, by repeated applications of Lemma 1:

Theorem 3 *If Algorithm* PARTITION *returns* PARTITIONING SUCCEEDED *on task system τ, then the resulting partitioning is* EDF*-feasible.*

Proof Sketch: Observe that Algorithm PARTITION 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. ■

Run-time complexity

In attempting to map task τ_i, observe that Algorithm PARTITION essentially evaluates, in Equations 5 and 6, 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 $m \leq n$.

5 Evaluation

As stated in Section 1, Algorithm PARTITION 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 Algorithm PARTITION. Indeed, this is to be expected since a simpler problem – partitioning collections of implicit-deadline sporadic tasks – 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 Algorithm PARTITION. Specifically, we derive some properties (Theorem 4 and Corollary 1) of Algorithm PARTITION, which characterize its performance. We would like to stress that *these properties are not intended to be used as feasibility tests to determine whether Algorithm* PARTITION *would successfully schedule a given sporadic task system* – since Algorithm PARTITION 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 Algorithm PARTITION 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 Algorithm PARTITION 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[1]:

$$\delta_{\max} \stackrel{\text{def}}{=} \max_{i=1}^{n} (e_i/d_i) \quad (10)$$

$$\delta_{\text{sum}} \stackrel{\text{def}}{=} \max_{t>0} \left(\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, t)}{t}\right) \quad (11)$$

$$u_{\max} \stackrel{\text{def}}{=} \max_{i=1}^{n} (u_i) \quad (12)$$

$$u_{\text{sum}} \stackrel{\text{def}}{=} \sum_{j=1}^{n} u_j \quad (13)$$

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. Lemma 2 follows immediately.

Lemma 2 *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*

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

[1]Observe that $\delta_{\max}$ defined below is the same as $\lambda_{\max}$ as defined in Section 3; however, δ_{sum} and λ_{sum} are not the same.

and

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

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 $m\xi \geq u_{\text{sum}}$ 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 $m\xi \geq \delta_{\text{sum}}$ 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 $(\sum_{j=1}^{n} \text{DBF}(\tau_j, t_o))/t_o = \delta_{\text{sum}}$. 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 2 above specifies necessary conditions for Algorithm PARTITION (or indeed, any algorithm) to successfully partition a sporadic task system; Theorem 4 below specifies a *sufficient* condition. But first, a technical lemma that will be used in the proof of Theorem 4.

Lemma 3 *Suppose that Algorithm* PARTITION *is attempting to schedule task system τ on a platform comprised of unit-capacity processors.*

C1: *If $u_{\text{sum}} \leq 1$, then Condition 6 is always satisfied.*

C2: *If $\delta_{\text{sum}} \leq \frac{1}{2}$, then Condition 5 is always satisfied.*

Consequently, any τ satisfying $u_{\text{sum}} \leq 1$ and $\delta_{\text{sum}} \leq \frac{1}{2}$ is successfully partitioned (on any number of processors ≥ 1.)

Proof: The proof of C1 is straightforward, since violating Condition 6 requires that $(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_k)$ exceed 1.

To see why C2 holds as well, 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 Inequality 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 5. But, violating Condition 5 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 both $u_{\text{sum}} \leq 1$ and $\delta_{\text{sum}} \leq \frac{1}{2}$ is successfully scheduled by Algorithm PARTITION. We now describe, in Theorem 4, what happens when one or both these conditions are not satisfied.

Theorem 4 *Any sporadic task system τ is successfully scheduled by Algorithm* PARTITION *on m unit-capacity processors, for any*

$$m \geq \left(\frac{2\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}} + \frac{u_{\text{sum}} - u_{\max}}{1 - u_{\max}} \right) \qquad (14)$$

Proof: Let us suppose that Algorithm PARTITION 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 and let Π_1 denote the m_1 processors upon which this mapping fails because Condition 5 is not satisfied (hence for the remaining $m_2 \stackrel{\text{def}}{=} (m - m_1)$ processors, denoted Π_2, Condition 5 is satisfied but Condition 6 is not).

By Lemma 3 above, m_1 will equal 0 if $\delta_{\text{sum}} \leq \frac{1}{2}$, while m_2 will equal 0 if $u_{\text{sum}} \leq 1$. Since we are assuming that the partitioning fails, it is not possible that both $\delta_{\text{sum}} \leq \frac{1}{2}$ and $u_{\text{sum}} \leq 1$ 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.

Since τ_i fails the test of Condition 5 on each processor in Π_1, it must be the case that each processor $\pi_\ell \in \Pi_1$ 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 in $\tau(\Pi_1)$ is a subset of the tasks in τ, we obtain

$$\begin{aligned} & \sum_{j=1}^{n} \text{DBF}^*(\tau_j, d_i) > m_1(d_i - e_i) + e_i \\ \Rightarrow \quad & \text{(By Inequality 3)} \\ & 2\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) > m_1(d_i - e_i) + e_i \\ \Rightarrow \quad & \frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} > \frac{m_1}{2}(1 - \frac{e_i}{d_i}) + \frac{e_i}{2d_i} \qquad (15) \end{aligned}$$

By definition of δ_{sum} (Equation 11)

$$\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} \leq \delta_{\text{sum}} \qquad (16)$$

Chaining Inequalities 15 and 16 above, we obtain

$$\begin{aligned} & \frac{m_1}{2}(1 - \frac{e_i}{d_i}) + \frac{e_i}{2d_i} < \delta_{\text{sum}} \\ \Rightarrow \quad & m_1 < \frac{2\delta_{\text{sum}} - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} \qquad (17) \end{aligned}$$

Now let us consider the processors in Π_2. Since none of these processors satisfy Condition 6 for task τ_i, it is 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 in Π_2, and noting that the tasks in $\tau(\Pi_2)$ is a subset of the tasks in τ, we obtain the following upper bound on the value of m_2:

$$(1 - u_i)m_2 + u_i < \sum_{j=1}^{n} u_j$$
$$\Rightarrow \qquad m_2 < \frac{u_{\text{sum}} - u_i}{1 - u_i} \tag{18}$$

To recap the proof thus far: we have assumed that our partitioning algorithm fails to place task τ_i on any processor, because Condition 5 fails on m_1 processors while Condition 2 fails on (at least) $m_2 = m - m_1$ processors. Inequalities 17 and 18 above represent upper bounds on values for m_1 and m_2, in terms of the parameters of tasks in τ.

We now consider three separate cases.

Case (i): ($\delta_{\text{sum}} > \frac{1}{2}$ and $u_{\text{sum}} \leq 1$). As stated in Lemma 3 (C1), Condition 6 is never violated in this case, and m_2 is consequently equal to zero. From this and Inequality 17, we therefore have

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

In order for Algorithm PARTITION 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, this implies that

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

which certainly holds for any m satisfying the statement of the theorem (Inequality 14).

Case (ii): ($\delta_{\text{sum}} \leq \frac{1}{2}$ and $u_{\text{sum}} > 1$). As stated in Lemma 3 (C2), Condition 5 is never violated in this case, and m_1 is consequently equal to zero. From this and Inequality 18, we therefore have

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

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

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

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

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

which once again holds for any m satisfying the statement of the theorem (Inequality 14).

Case (iii): ($u_{\text{sum}} > 1$ and $\delta_{\text{sum}} > \frac{1}{2}$). In this case, both m_1 and m_2 may be non-zero. From $m_1 + m_2 = m$ and Inequality 18, we may conclude that

$$m_1 > m - \frac{u_{\text{sum}} - u_i}{1 - u_i} \tag{19}$$

For Inequalities 19 and 17 to both be satisfied, we must have

$$\frac{2\delta_{\text{sum}} - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} > m - \frac{u_{\text{sum}} - u_i}{1 - u_i}$$
$$\Rightarrow \qquad m < \frac{2\delta_{\text{sum}} - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} + \frac{u_{\text{sum}} - u_i}{1 - u_i} \tag{20}$$

Hence for Algorithm PARTITION to successfully schedule τ, it is sufficient that the negation of Inequality 20 hold:

$$m \geq \left(\frac{2\delta_{\text{sum}} - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} + \frac{u_{\text{sum}} - u_i}{1 - u_i} \right) \tag{21}$$

Observe that the right hand-side of Inequality 21 is maximized when u_i and $\frac{e_i}{d_i}$ are both as large as possible; by Inequalities 10 and 11, these are defined to be $u_{\max}$ and $\delta_{\max}$ respectively. We hence get Inequality 14:

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

as a sufficient condition for τ to be successfully scheduled by Algorithm PARTITION. ■

Using Theorem 4 above, we now present a resource-augmentation result concerning Algorithm PARTITION.

Corollary 1 *Algorithm* PARTITION *makes the following performance guarantee:* if a sporadic task system is feasible on m identical processors each of a particular computing capacity, then Algorithm PARTITION 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.

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 Algorithm PARTITION on m unit-capacity processors, for all values of $\xi \leq \frac{m}{4m-2}$.

Since τ is feasible on m ξ-speed processors, it follows from Lemma 2 that the tasks in τ satisfy the following properties:

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

By substituting these in Equation 14, we get

$$\begin{aligned} m &\geq \frac{2\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}} + \frac{u_{\text{sum}} - u_{\max}}{1 - u_{\max}} \\ \Leftarrow \quad m &\geq \frac{2m\xi - \xi}{1 - \xi} + \frac{m\xi - \xi}{1 - \xi} \\ \equiv \quad \xi &\leq \frac{m}{4m - 2} \end{aligned}$$

which is as claimed in Corollary 1. ■

Extensions, and Discussion

In Section 2, we defined *constrained* sporadic task systems to be ones in which it is guaranteed that all tasks $\tau_i = (e_i, d_i, p_i)$ in the system have their relative deadline parameters be no larger than their period parameters: $d_i \leq p_i$. Such constrained systems have been the subject of much study in uniprocessor scheduling theory; below, we summarize some of the results we have obtained concerning the multiprocessor partitioned scheduling of constrained sporadic task systems. We omit proofs here due to space considerations; proofs will be provided in a more complete version of this report, currently under preparation.

Theorem 5 *Consider any constrained sporadic task system τ that is (global or partitioned) feasible on m identical processors.*

1. *Sporadic task system τ can be partitioned, in polynomial time, to always meet all deadlines on m identical processors in which the individual processors are $(3 - \frac{1}{m})$ times as fast as in the original system, when* EDF *is used to schedule each processor during run-time.*

2. *Sporadic task system τ can be partitioned, in pseudo-polynomial time, to always meet all deadlines on m identical processors in which the individual processors are $(3 - \frac{1}{m})$ times as fast as in the original system, when the* **deadline monotonic** *scheduling algorithm (DM) [8] is used to schedule each processor during run-time.*

3. *Sporadic task system τ can be partitioned, in polynomial time, to always meet all deadlines on m identical processors in which the individual processors are $(6 - \frac{2}{m})$ times as fast as in the original system, when DM is used to schedule each processor during run-time.*

We reiterate that the results in Corollary 1 and Theorem 5 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.

6 Context and Related Work

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 implicit-deadline ("$d_i = p_i$") 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 [9]) to arbitrary systems 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.

Currently not too much seems to be known about the multiprocessor scheduling of sporadic task systems. The only other papers that we are aware of that specifically address this subject are [2, 3, 5]; in contrast to our work here on partitioned scheduling, all of these papers focus on the *global* paradigm of multiprocessor scheduling. and present different sufficient conditions for guaranteeing that a given multiprocessor sporadic task system is guaranteed to always meet all deadlines during run-time when scheduled using global EDF.

7 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.

References

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

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

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

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

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

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

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

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

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

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

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

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