The Partitioned, Static-Priority Scheduling of Sporadic Real-Time Tasks with Constrained Deadlines on Multiprocessor Platforms*

Nathan Fisher and Sanjoy Baruah

Department of Computer Science, The University of North Carolina at Chapel Hill
{fishern, baruah}@cs.unc.edu

Abstract. We consider the partitioned scheduling of sporadic, hard-real-time tasks on a multiprocessor platform with static-priority scheduling policies. Most previous work on the static-priority scheduling of sporadic tasks upon multiprocessors has assumed implicit deadlines (i.e. a task's relative deadline is equal to its period). We relax the equality constraint on a task's deadline and consider task systems with constrained deadlines (i.e. relative deadlines are at most periods). In particular, we consider the *first-fit decreasing* partitioning algorithm. Since the partitioning problem is easily seen to be NP-hard in the strong sense, this algorithm is unlikely to be optimal. We quantitatively characterize the partitioning algorithm's worst-case performance in terms of *resource augmentation*.

1 Introduction

Over the years, the sporadic task model [23] has proven remarkably useful for the modelling of recurring processes in hard-real-time systems where the release times of jobs are not known *a priori*. 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 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. [9]) that a sporadic task system is feasible on a preemptive uniprocessor if and only if all deadlines can be

* Supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825).

J.H. Anderson, G. Prencipe, and R. Wattenhofer (Eds.): OPODIS 2005, LNCS 3974, pp. 291–305, 2006.

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).

Feasibility of a sporadic task system under static-priority scheduling has a more restrictive definition. In static-priority systems, each task is assigned a distinct priority, and all jobs of a task execute at the task's priority. A task system is said to be feasible *with respect to a static-priority scheduling algorithm* if the resulting priority assignment results in all deadlines being met under all legal combinations of job-arrival sequences. For *constrained-deadline systems* (i.e. each task has $d_i \leq p_i$), a sporadic system is feasible on a preemptive uniprocessor if and only if it can be scheduled according to the Deadline-Monotonic algorithm (DM) [19]; DM assigns priority to task according to on the inverse of its relative deadline. Several exact algorithms for feasibility-analysis of static-priority task systems (both unconstrained- and constrained-deadline) upon uniprocessor platforms have been developed [16, 1, 17].

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. Leung and Whitehead [19] showed that determining the feasibility of a task system under the partitioned paradigm is NP-hard. Under global scheduling, it is permitted that a job that has previously been preempted on one processor can 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 theoretical 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]. Oh [25] gives a performance bound on the *first-fit decreasing* heuristic using exact feasibility tests for deadline-monotonic scheduling. A significant portion of prior research has focused on deriving partitioning algorithms based on *utilization bounds*. A utilization bound represents the highest utilization in which any task system possessing a utilization at most the bound is guaranteed to be feasible on the multiprocessor system. Oh and Baker [24] derive a sufficient utilization test for feasibility of a task system that is partitioned scheduled using DM algorithm and assuming an implicit-deadline task system. Burchard, et al. [12] present utilization conditions for partitioned DM based on tightened uniprocessor utilization bounds derived from Liu and Layland [20].

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 *arbi-*

[1] Notable and important exceptions are the recent work of Baker [4, 5, 6, 7], Baruah and Fisher [8], and Bertogna et al. [10] which consider systems of sporadic tasks with $d_i \neq p_i$.

trary 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 [20]) to arbitrary systems (as represented in, e.g. [23, 18, 19, 17, 2] etc.[2], 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.

Organization. In this paper, we report our findings concerning the preemptive multiprocessor scheduling of constrained-deadline sporadic real-time systems under the partitioned paradigm. The remainder of the paper is organized as follows. In Section 2, we formally specify the task model, and define the *request-bound function* and *demand-bound function* as characterizations of the maximum amount of execution time that has been requested by a task over a time interval of a given length. In Section 3, we position our finds within a larger context of multiprocessor real-time scheduling theory, and compare and contrast our results with related research. In Section 4, we describe an exact feasibility test for *uni*processor static-priority scheduling. In Section 5, we describe the *first-fit decreasing* algorithm for the partitioning of sporadic tasks upon a multiprocessor system. In Section 6, we theoretically evaluate the conditions under which first-fit decreasing is guaranteed to be able to successfully partition a task system on a multiprocessor platform. In Section 7, we state some future directions of our research.

2 Task/Machine Model and Definitions

In the sporadic task 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 (historically, referred to as the *period* of a task). 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.

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, $1 \leq i \leq n$. 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:

[2] This is merely a small sample, and by no means an exhaustive list of citations of important and influential papers.

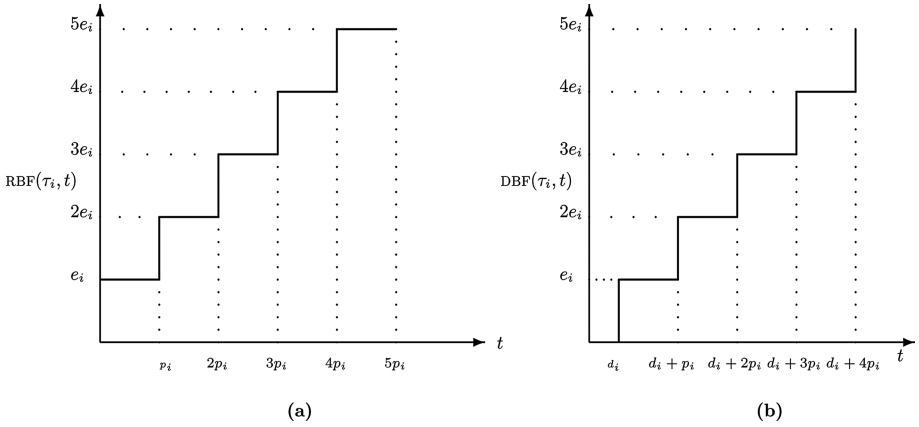

Fig. 1. **(a)** denotes a plot of $\text{RBF}(\tau_i, t)$ as a function of t. **(b)** denotes a plot of $\text{DBF}(\tau_i, t)$ as a function of t.

- 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 results we obtain in this work is for constrained task systems. Therefore, we will assume that a given task system, τ, is constrained, unless otherwise specified.

2.1 The Request-Bound Function

For any sporadic task τ_i and any real number $t \geq 0$, the *request-bound function* $\text{RBF}(\tau_i, t)$ is the largest cumulative execution requirement of all jobs that can be generated by τ_i to have their arrival times within a contiguous interval of length t. Every time a task τ_i releases a job, e_i additional units of processor time are requested. The following function provides an upper bound on the total execution time requested by task τ_i at time t (i.e. the scenario where a task releases jobs as soon as legally possible):

$$\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{p_i} \right\rceil e_i. \tag{1}$$

Figure 1a shows an example of a RBF. Notice that the "step" function, $\text{RBF}(\tau_i, t)$ increases by e_i units every p_i time units.

To determine the response-time for the first job of task τ_i on a preemptive uniprocessor, we must consider execution requests of τ_i and all jobs of tasks which may preempt τ_i. We define the following *cumulative request-bound function*

based on RBF. Let $\mathbf{T}_{H_i}$ be the set of tasks with priority greater than τ_i. Then, the cumulative request-bound function is defined as:

$$W_i(t) \stackrel{\text{def}}{=} e_i + \sum_{\tau_j \in \mathbf{T}_{H_i}} \text{RBF}(\tau_j, t). \tag{2}$$

The cumulative request-bound function $W_i(t)$ is simply the total execution requests of all tasks of higher priority than τ_i over the interval $[0, t)$, and the execution request of one job of τ_i.

2.2 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 (see Figure 1b for a visual example). It has been shown [9] 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 (3) below follows directly [9]:

$$\text{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \max\left(0, \left(\left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) \times e_i\right). \tag{3}$$

2.3 Relationship Between RBF and DBF

For all task systems, $\text{RBF}(\tau_i, t)$ exceeds or equals $\text{DBF}(\tau_i, t)$ for all $t > 0$. In a constrained task system τ, an important property regarding the ratio between RBF and DBF holds. For each task $\tau_i \in \tau$, the ratio between the RBF and DBF,for all $t \geq d_i$, is bounded by two. The following lemma formally states this observation:

Lemma 1. *If $d_i \leq p_i$, then for all $t \in [d_i, \infty)$, $\text{RBF}(\tau_i, t) \leq 2 \cdot \text{DBF}(\tau_i, t)$.*

Proof: Recall that $d_i \leq p_i$; thus, $\frac{d_i}{p_i} \leq 1$. So, for $t \geq d_i$:

$$\left\lceil \frac{t}{p_i} \right\rceil - \left\lfloor \frac{t - d_i}{p_i} \right\rfloor \leq \left\lceil \frac{t}{p_i} \right\rceil - \left\lfloor \frac{t}{p_i} \right\rfloor + 1 \leq 2. \tag{4}$$

Multiplying both sides of Equation 4 by e_i, we get:

$$\begin{array}{ll} & \left\lceil \frac{t}{p_i} \right\rceil e_i - \left\lfloor \frac{t-d_i}{p_i} \right\rfloor e_i \leq 2e_i \\ \Rightarrow & \left\lceil \frac{t}{p_i} \right\rceil e_i \qquad\qquad\quad \leq \left(\left\lfloor \frac{t-d_i}{p_i} \right\rfloor + 1\right) e_i + e_i \\ \Rightarrow & \text{RBF}(\tau_i, t) \qquad\quad \leq \text{DBF}(\tau_i, t) + e_i \quad \text{(by definition of RBF and DBF)} \\ \Rightarrow & \text{RBF}(\tau_i, t) \qquad\quad \leq 2 \cdot \text{DBF}(\tau_i, t). \end{array}$$

The last step follows because for $t \geq d_i$, $\text{DBF}(\tau_i, t) \geq e_i$. □

Lemma 1 will be useful in Section 6 for deriving quantitative results for the first-fit decreasing partitioning algorithm.

3 Context and Related Work

Given the specifications of a multiprocessor sporadic task system, **feasibility analysis** is the process of determining whether it is possible for the system to meet all timing constraints under all legal combinations of job arrivals. For **implicit-deadline** systems, partitioned feasibility-analysis can be transformed to a bin-packing problem [14] and shown to be NP-hard in the strong sense; sufficient feasibility tests for various bin-packing heuristics have recently been obtained [22, 21]. The **constrained** and **arbitrary** task models are generalizations of the implicit-deadline model; therefore, the intractability result (feasibility analysis being NP-hard in the strong sense) continues to hold. To our knowledge, there have been no prior non-trivial positive theoretical results (other than [8]) 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 [22, 21, 24].

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?* 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.)

The technique of **resource augmentation** [15, 26] is sometimes used 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)?

In prior work [8], we developed a partitioning algorithm for constrained and arbitrary task systems using dynamic-priority scheduling policies, and analyzed its performance from the perspective of resource augmentation. Our results, in this paper, are with respect to *partitioned* static-priority scheduling of sporadic, *constrained* task systems. We will derive the following resource-augmentation bound from these results (see Corollary 2):

> If a system of sporadic tasks is feasible under the global paradigm (and consequently, under the partitioned paradigm as well) on m identical processors, then this system of sporadic tasks is partitioned by the first-fit decreasing algorithm on m identical processors in which the individual processors are $(3 - \frac{1}{m})$ times as fast as in the original system, such that each partition is uniprocessor DM-schedulable.

4 Uniprocessor Exact-Feasibility Test

In this section, we present uniprocessor exact-feasibility tests for a sporadic task set, τ, in a static-priority system. We will use these tests as the basis for the partitioning algorithm defined in Section 5 and the sufficient feasibility conditions developed in Section 6.

For static-priority task systems with relative deadlines bounded by periods, Liu and Layland [20] showed that the *worst-case* response time for a job of task τ_i occurs when all tasks of priority greater than τ_i release a job simultaneously with τ_i under the synchronous arrival sequence. In a sporadic task system with constrained deadlines, it is necessary and sufficient to only check the worst-case response time of the first job of each task. If the worst-case response time of the first job of task τ_i is at most its relative deadline, then τ_i is schedulable; else, it is not schedulable. A task system τ is feasible on a uniprocessor *if and only if* the first job of each task τ_i has a worst-case response time at most d_i.

Audsley *et al.* [3] presented an exact feasibility test for task τ_i using DM: a task always meets all deadlines on a preemptive uniprocessor if and only if there exists a fixed point, t, of $W_i(t)$ such that t occurs before τ_i's deadline . The following theorem restates their test:

Theorem 1 (from [3]). *In a constrained sporadic task system, task τ_i always meets all deadlines using* DM *on a preemptive uniprocessor if and only if $\exists t \in (0, d_i]$ such that $W_i(t) \leq t$.* □

5 A Partitioning Algorithm

Given sporadic task system τ comprised of n tasks $\tau_1, \tau_2, \ldots \tau_n$, and a platform comprised of m unit-capacity processors $\pi_1, \pi_2, \ldots, \pi_m$, we now describe an algorithm for partitioning the tasks in τ among the m processors. With no loss of generality, let us assume that *tasks are indexed according to non-decreasing order of their relative deadline parameter* (i.e., $d_i \leq d_{i+1}$ for all i, $1 \leq i < n$). The algorithm 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. The algorithm for doing this is a variant of the *First-Fit Decreasing* [13, 25] algorithm for bin-packing, and is as follows (see Figure 2 for a pseudo-code representation of FFD-DM). 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 order, we will assign task τ_i to the first processor π_ℓ, $1 \leq \ell \leq m$, that satisfies the following condition:

$$\exists t \in (0, d_i] :: W_{i,\pi_\ell}(t) \leq t, \tag{5}$$

where

$$W_{i,\pi_\ell}(t) \stackrel{\text{def}}{=} e_i + \sum_{\tau_j \in \mathbf{T}_{H_i} \cap \tau(\pi_\ell)} \text{RBF}(\tau_j, t). \tag{6}$$

```
FFD-DM(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 parame-
      ter: d_i ≤ d_{i+1} for all i.) τ(π_ℓ) denotes the tasks assigned to processor π_ℓ;
      initially, τ(π_ℓ) ← ∅ for all ℓ.
1   for i ← 1 to n
    ▷ i ranges over the tasks, which are indexed by non-decreasing value of
    the deadline parameter
2           for ℓ ← 1 to m
            ▷ ℓ ranges over the processors, considered in order
3                   if τ_i satisfies Condition 5 on processor π_ℓ then
                            ▷ assign τ_i to π_ℓ; proceed to next task
4                           τ(π_ℓ) ← τ(π_ℓ) ⋃ {τ_i}
5                           goto line 7
6           end (of inner for loop)
7           if (ℓ > m) return PARTITIONING FAILED
8   end (of outer for loop)
9   return PARTITIONING SUCCEEDED
```

Fig. 2. Pseudo-code for partitioning algorithm

If no such π_ℓ 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_ℓ, the 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* DM*-feasible on that processor and the algorithm above assigns τ_i to processor π_ℓ (according to Condition 5), then the tasks assigned to each processor (including processor π_ℓ) remain feasible on that processor.*

Proof: Observe that the feasibility of processors other than processor π_ℓ is not affected by the assignment of task τ_i to processor π_ℓ. It remains to demonstrate that, if the tasks assigned to π_ℓ were feasible on π_ℓ prior to the assignment of τ_i and Condition 5 is satisfied, then the tasks on τ_ℓ remain feasible after adding τ_i.

Observe that the following conditions held prior to adding τ_i to processor π_ℓ (due to Condition 5 of partitioning algorithm):

$$\forall \tau_j \in \mathbf{T}_{H_i} \cap \tau(\pi_\ell) : (\exists t \in (0, d_j] :: W_{j,\pi_\ell}(t) \leq t) \,. \tag{7}$$

In a static-priority, uniprocessor system adding a lower-priority task τ_i does not affect the $W_{j,\pi_\ell}(t)$ function for all $\tau_j \in \mathbf{T}_{H_i} \cap \tau(\pi_\ell)$. Therefore, Condition 7 continues to hold after the assignment of τ_i to processor π_ℓ. By Theorem 1, all tasks τ_j previously assigned to processor π_ℓ remain feasible. □

The correctness of the partitioning algorithm follows, by repeated applications of Lemma 2:

Theorem 2. *If the* FFD-DM *partitioning algorithm returns* PARTITIONING SUCCEEDED *on constrained task system τ, then the resulting partitioning is* DM*-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_i, each processor is trivially DM-feasible. It follows from Lemma 2 that all processors remain DM-feasible after each task assignment as well. Hence, all processors are DM-feasible once all tasks in τ have been assigned. □

5.1 Run-Time Complexity

It may appear that a potentially infinite number of values of t must be checked in order to ensure that Equation 5 is satisfied. However, Lehoczky, et al [16] showed that it suffices to check only the values in the set,

$$S_{i,\pi_\ell} \stackrel{\text{def}}{=} \left\{ t = bp_j : \tau_j \in \tau(\pi_\ell) \cap \mathbf{T}_{H_i}, b = 1, \ldots, \left\lfloor \frac{d_i}{p_j} \right\rfloor \right\} \tag{8}$$

to determine whether τ_i fits on π_ℓ. We may have to check all m processors to determine whether τ_i "fits" on the multiprocessor platform. Thus, in the worst-case, we need to check all the points in

$$S_i \stackrel{\text{def}}{=} S_{i,\pi_1} \cup \ldots \cup S_{i,\pi_m} = \left\{ t = bp_j : j = 1, \ldots, i; b = 1, \ldots, \left\lfloor \frac{d_i}{p_j} \right\rfloor \right\}. \tag{9}$$

The number of elements in S_i is $\mathcal{O}(i \left\lfloor \frac{d_i}{p_1} \right\rfloor)$. Therefore, to check Equation 5 for a task τ_i requires $\mathcal{O}(i \left\lfloor \frac{d_i}{p_1} \right\rfloor + m)$. The time complexity for testing the feasibility of the entire task set τ is $\mathcal{O}(n^2 \left\lfloor \frac{d_n}{p_1} \right\rfloor)$ under the reasonable assumption that $m \leq n$. Since the time complexity of the (exact-test-based) partitioning algorithm depends upon the period and relative deadline parameters of τ, the algorithm runs in *pseudo-polynomial* time. The run-time of determining whether Equation 5 is satisfied can be further reduced by considering the tunable feasibility test of Bini and Buttazzo [11].

6 Evaluation

As stated in Section 1, the first-fit decreasing 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 FFD-DM. 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 the FFD-DM algorithm runs in pseudo-polynomial

time. In this section, we offer a quantitative evaluation of the efficacy the algorithm. Specifically, we derive some properties (Theorem 3 and Corollary 2) of the FFD-DM 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* FFD-DM *would successfully schedule a given sporadic task system*. Rather, these properties are intended to provide a quantitative measure of how effective FFD-DM partitioning is *vis a vis* the performance of an optimal scheduler. For an empirical evaluation of the FFD-DM algorithm considered in this paper, we refer the reader to [7].

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

$$\delta_{\max}(\tau) \stackrel{\text{def}}{=} \max_{i=1}^{n} \left(e_i / d_i \right), \tag{10}$$

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

Lemma 3. *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*

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

and

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

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

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

Lemma 4. *Suppose that the* FFD-DM *partitioning algorithm is attempting to schedule a constrained sporadic task system τ on a platform comprised of unit-capacity processors.*

If $\delta_{\text{sum}}(\tau) \leq \frac{1}{2}$, then Equation 5 is always satisfied.

Proof: 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 $t_o \geq 0$. For any task $\tau_i \in \tau$, $\sum_{j=1}^{i} \text{DBF}(\tau_j, t_o) \leq \frac{t_o}{2}$. By Lemma 1, this in turn implies that $\sum_{j=1}^{i} \text{RBF}(\tau_j, t_o) \leq t_o$ for all $t_o \geq d_i$; notice that $\text{RBF}(\tau_i, d_i) \geq e_i$.

Thus, for all $\tau_i \in \tau$, $\sum_{j=1}^{i-1} \text{RBF}(\tau_j, d_i) + e_i \leq d_i$. This implies for any processor π_ℓ:

$$\exists t_o \in (0, d_i] :: W_{i,\pi_\ell}(t_o) \leq t_o.$$

Therefore, Equation 5 is satisfied. □

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

Thus, any sporadic task system satisfying $\delta_{\text{sum}}(\tau) \leq \frac{1}{2}$ is successfully scheduled by the FFD-DM. We now describe, in Theorem 3, what happens when this condition is not satisfied.

Theorem 3. *Any constrained sporadic task system τ is successfully scheduled by* FFD-DM *on m unit-capacity processors, for any*

$$m \geq \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \delta_{\max}(\tau)}. \tag{12}$$

Proof: The proof is by contradiction. Assume that m satisfies the antecedent of the theorem, but cannot schedule τ on m processors by FFD-DM. Then there exists a task τ_i which does not fit on any processor according Equation 5. It must be the case (by Theorem 1) that each such processor π_ℓ satisfies

$$\begin{aligned} & W_{i,\pi_\ell}(d_i) && > d_i \\ \Rightarrow & \textstyle\sum_{\tau_j \in \tau(\pi_\ell)} \text{RBF}(\tau_j, d_i) + e_i && > d_i \\ \Rightarrow & \textstyle\sum_{\tau_j \in \tau(\pi_\ell)} 2 \cdot \text{DBF}(\tau_j, d_i) && > d_i - e_i \text{ (according to Lemma 1).} \end{aligned}$$

Observe that $\text{DBF}(\tau_i, d_i) = e_i$ and $\text{DBF}(\tau_j, d_i) = 0$ for all $j > i$. Summing over all m such processors and noting that the tasks on these processors is a subset of the tasks in τ, we obtain

$$\begin{aligned} & 2\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) > m(d_i - e_i) + e_i \\ \Rightarrow \; & \frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} > \frac{m}{2}(1 - \frac{e_i}{d_i}) + \frac{e_i}{2d_i}. \end{aligned} \tag{13}$$

By definition of $\delta_{\text{sum}}(\tau)$ (Equation 11)

$$\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} \leq \delta_{\text{sum}}(\tau). \tag{14}$$

Chaining Inequalities 13 and 14 above, we obtain

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

By Corollary 1, it is necessary that $\delta_{\text{sum}}(\tau) > \frac{1}{2}$ hold. Since $\delta_{\max}(\tau) \leq 1$ (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

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

which contradicts Inequality 12 above. □

The technique of **resource augmentation** may be used to quantify the "goodness" (or otherwise) of an algorithm for solving problems for which optimal algorithms are either impossible in practice (e.g., because optimal decisions require knowledge of future events), or computationally intractable. In this technique, the performance of the algorithm being discussed is compared with that of a hypothetical optimal one, under the assumption that the algorithm under discussion has access to *more resources* than the optimal algorithm. Using Theorem 3 above, we now present such a result concerning FFD-DM.

Theorem 4. FFD-DM *makes the following performance guarantees: if a constrained sporadic task system is feasible on m_o identical processors each of a particular computing capacity, then* FFD-DM *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.*

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. Since τ is feasible on m_o ξ-speed processors, it follows from Lemma 3 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 we attempt to schedule τ using FFD-DM on $m \geq \frac{2m_o\xi - \xi}{1-\xi}$ unit-capacity processors. By substituting the inequalities above, we satisfy the condition of Theorem 3:

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

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

By setting $m_o \leftarrow m$ in the statement of Theorem 4 above, we immediately have the following corollary.

Corollary 2. FFD-DM *makes the following performance guarantees:*

If a constrained sporadic task system is feasible on m identical processors each of a particular computing capacity, then FFD-DM *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.* □

7 Conclusions and Future Work

Prior theoretical research [12, 24] on the static-priority scheduling of sporadic task systems upon partitioned multiprocessors has assumed that a task's deadline is equal to its period parameter. In this work, we have relaxed this constraint, and have allowed a task's period parameter to exceed its deadline parameter. We consider the scheduling of such sporadic task systems upon preemptive multiprocessor platforms, under the partitioned scheduling paradigm. To this end, we consider the first-fit decreasing partitioning algorithm using an exact-uniprocessor feasibility tests as a condition for a task "fitting" on a processor. We have proven the correctness of the partitioning algorithm, and have characterized the conditions under which the algorithm correctly partitions a task system. In particular, we have shown that the algorithm can partition a globally feasible task system by augmenting the speed of each processor by a multiplicative factor.

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.

We have not considered in this paper arbitrary sporadic task systems (i.e. task systems where there is no restriction between a task's relative deadline and its period parameter). Unfortunately, preliminary research indicates that the sufficient feasibility tests from this paper do not directly generalize for arbitrary sporadic tasks. Formally, the reason the results do not generalize is due to the fact that Lemma 1 does not hold for arbitrary task systems. We are currently working on devising sufficient feasibility tests for arbitrary sporadic task systems scheduled under the partitioned multiprocessor paradigm.

References

1. Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal 8*, 5 (1993), 285–292.
2. Audsley, N., Burns, A., and Wellings, A. Deadline monotonic scheduling theory and application. *Control Engineering Practice 1*, 1 (1993), 71–78.

3. Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, A. J. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software* (Atlanta, May 1991), pp. 127–132.
4. 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.
5. Baker, T. P. An analysis of deadline-monotonic schedulability on a multiprocessor. Tech. Rep. TR-030201, Department of Computer Science, Florida State University, 2003.
6. Baker, T. P. An analysis of EDF schedulability on a multiprocessor. Tech. Rep. TR-030202, Department of Computer Science, Florida State University, 2003.
7. Baker, T. P. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Tech. Rep. TR-050601, Department of Computer Science, Florida State University, 2005.
8. Baruah, S., and Fisher, N. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the 26th Real-Time Systems Symposium* (Miami, Florida, December 2005), IEEE Computer Society Press.
9. Baruah, S., Mok, A., and Rosier, L. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium* (Orlando, Florida, 1990), IEEE Computer Society Press, pp. 182–190.
10. Bertogna, M., Cirinei, M., and Lipari, G. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Palma de Mallorca, Balearic Islands, Spain, July 2005), IEEE Computer Society Press, pp. 209–218.
11. Bini, E., and Buttazzo, G. The space of rate monotonic schedulability. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium* (Austin, Texas, December 2002), IEEE Computer Society Press, pp. 169–178.
12. Burchard, A., Liebeherr, J., Oh, Y., and Son, S. H. Assigning real-time tasks to homogeneous multiprocessor systems. *IEEE Transactions on Computers 44*, 12 (December 1995), 1429–1442.
13. Johnson, D. Fast algorithms for bin packing. *Journal of Computer and Systems Science 8*, 3 (1974), 272–314.
14. Johnson, D. S. *Near-optimal Bin Packing Algorithms*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1973.
15. Kalyanasundaram, B., and Pruhs, K. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)* (Los Alamitos, Oct. 1995), IEEE Computer Society Press, pp. 214–223.
16. Lehoczky, J., Sha, L., and Ding, Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989* (Santa Monica, California, USA, Dec. 1989), IEEE Computer Society Press, pp. 166–171.
17. Lehoczky, J. P. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium* (Dec. 1990), pp. 201–209.
18. Leung, J., and Merrill, M. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters 11* (1980), 115–118.
19. Leung, J., and Whitehead, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2* (1982), 237–250.
20. Liu, C., and Layland, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (1973), 46–61.

21. 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.
22. 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.
23. Mok, A. K. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.
24. Oh, D.-I., and Baker, T. P. Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems: The International Journal of Time-Critical Computing 15* (1998), 183–192.
25. Oh, Y. *The Design and Analysis of Scheduling Algorithms for Real-Time and Fault-Tolerant Computer Systems*. PhD thesis, Department of Computer Science, The University of Virginia, 1994.
26. Phillips, C. A., Stein, C., Torng, E., and Wein, J. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (El Paso, Texas, 4–6 May 1997), pp. 140–149.