The Partitioned Scheduling of Sporadic Tasks According to Static-Priorities*

Nathan Fisher Sanjoy Baruah
The University of North Carolina at Chapel Hill
Department of Computer Science, CB-3175
Chapel Hill, NC 27599-3175 USA
{fishern, baruah}@cs.unc.edu

Theodore P. Baker
Florida State University
Department of Computer Science
Tallahassee, FL 32306-4530
baker@cs.fsu.edu

Abstract

A polynomial-time algorithm is presented for partitioning a collection of sporadic tasks among the processors of an identical multiprocessor platform with static-priority scheduling on each individual processor. Since the partitioning problem is easily seen to be NP-hard in the strong sense, this algorithm is not optimal. A quantitative characterization of its worst-case performance is provided in terms of sufficient conditions and resource augmentation approximation bounds. The partitioning algorithm is also evaluated over randomly generated task systems.

1 Introduction

Uniprocessor scheduling has significantly benefitted from the trend towards the development of very general models of real-time computation. Originally, the *Liu and Layland task model* [17] imposed a strict set of assumptions and requirements on task systems. In the Liu and Layland model, a task must generate jobs at strict periodic intervals. In addition, each job is required to complete its execution prior to the arrival of the next job of the task (i.e. a job's relative deadline is equal to the task's period). An important generalization of the Liu and Layland model was the *sporadic task model* [18]. The sporadic task model allowed a job to have a different relative deadline from the task's period, and removed the requirement that a task generate jobs at strict periodic intervals. The generality of the sporadic task model and the theory that developed around it [18, 8, 15, 2] led to increased applicability of scheduling theory to practical uniprocessor real-time systems.

Considering the positive impact that the move from the Liu and Layland model to more general models of real-time computation had on the field of uniprocessor scheduling, it is natural to inquire whether similar benefits for multiprocessor scheduling may result from the study of sporadic task systems. However, researchers have only recently begun to consider the multiprocessor scheduling of general sporadic task systems. Our current research focuses on attempting to obtain scheduling theory results for the multiprocessor scheduling of sporadic task systems.

Feasibility-Analysis. An important scheduling theory concept is determining the *feasibility* of a task system on a specified platform. 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. [8]) 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).

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.

Multiprocessor Scheduling. 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 previ-

*This research has been supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, CCR-0309825, and CHR-EHS 0509131).

ously 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). Baker [3, 4] and Bertogna et al. [9, 10] considered the feasibility of general sporadic task system using global dynamic- and static-priority scheduling algorithms.

Partitioned scheduling of sporadic task systems has received some attention recently. Baruah and Fisher [7] developed and analyzed a polynomial-time algorithm for the partitioning of sporadic task systems using dynamic-priority scheduling algorithms. Fisher and Baruah [12] analyzed a pseudo-polynomial-time algorithm for partitioning sporadic task systems using static-priority scheduling algorithms. Baker [5] empirically evaluated the effectiveness of various partitioning schemes for sporadic task systems.

Our Contributions. The work presented in this paper makes the following theoretical and empirical contributions to partitioned multiprocessor scheduling:

- We present a polynomial-time partitioning algorithm for general sporadic task systems on an identical multiprocessor platform when static-priority scheduling policies are used on each processor.
- We evaluate our algorithm by providing sufficient conditions for feasibility, a *resource augmentation* approximation ratio, and simulation results. Our algorithm is the first polynomial-time algorithm for the multiprocessor static-priority scheduling of general sporadic tasks to provide a resource augmentation guarantee.
- We compare our results with known polynomial-time algorithms for partitioning Liu and Layland tasks. Specificially, we quantify the schedulability loss incurred by considering a more general task model.

Organization. In Section 2, we formally introduce the sporadic task model and identical multiprocessor platform model, and describe formal characterizations of task workload. In Section 3, we present our polynomial-time partitioning algorithm for sporadic task systems and prove its correctness. Section 4 evaluates the efficacy of the partitioning algorithm in terms of sufficient conditions for success and resource augmentation approximation bounds. In Section 5, we run simulations of our algorithm over a set of randomly generated task systems and present the results. In Section 6, we summarize the major conclusions and contributions of our paper.

2 Background

2.1 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 (also, called the task's *period*). In general, a sporadic task is subject to the trivial constraints that $e_i \leq p_i$ and $e_i \leq d_i$. The *utilization* of task τ_i represents the amount computational capacity required by τ_i on a single processor and is denoted by $u_i \stackrel{\text{def}}{=} e_i/p_i$.

We will assume that we are given a multiprocessor Π comprised of m identical processors, $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$. Let τ be a system of n sporadic tasks where $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and $\tau_i = (e_i, d_i, p_i)$ for all i, $1 \leq i \leq n$.

We may categorize sporadic task systems based on the relationship between the values of p_i and d_i for each $\tau_i \in \tau$. For the purposes of this paper, we consider three subclasses based on this relationship:

- **Implicit-deadline**: Each sporadic task $\tau_i \in \tau$ satisfies the constraint that $d_i = p_i$.
- **Constrained**: Each sporadic task $\tau_i \in \tau$ satisfies the constraint that $d_i \leq p_i$.
- **Arbitrary**: There is no restriction placed on the relationship between d_i and p_i.

2.2 Task-Workload Characterization

An important technique in feasibility-analysis is determining the maximum execution requirements that a sporadic task may generate over any interval of length t. If it is determined that the maximum execution requirements of each task over any interval length $t > 0$ can be fulfilled by the processing platform, then the task system is feasible on the processing platform.

For implicit-deadline systems, determining the maximum execution requirements of a task τ_i over an interval of length t is equivalent to calculating the expression $u_i \times t$. However, when the implicit-deadline assumption is removed, determining the maximum execution requirement of τ_i becomes more complex. Section 2.2.1 describes the *demand-bound function* (DBF), useful for characterizing the workload of a task in dynamic-priority scheduling. Section 2.2.2 describes the *request-bound function* (RBF), useful for characterizing the workload of a task in static-priority scheduling. Additionally, an approximation to RBF is described. Section 2.2.3 makes an observation about the relationship between the demand-bound function and the approximate request-bound function.

2.2.1 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 [8] 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 [8]:

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

2.2.2 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):

$$\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{p_i} \right\rceil e_i. \quad (2)$$

Fisher and Baruah [13] proposed a method for approximating RBF; the following function can be obtained by applying the approximation technique:

$$\text{RBF}^*(\tau_i, t) \stackrel{\text{def}}{=} e_i + u_i \times t. \quad (3)$$

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 RBF^* (Equation 3 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 upon its arrival, and that τ_i be assigned an additional $u_i \times \Delta t$ of execution over time-interval $[t, t + \Delta t)$, for all instants t after the arrival of the first job, and for arbitrarily small positive Δt.

2.2.3 Relationship Between RBF^* and DBF

The following observation will be important in quantitatively evaluating the partitioning algorithm presented in the next section. The next lemma essentially provides an upper bound on $\text{RBF}^*(\tau_i, t)$ in terms of τ_i's utilization and demand-bound function.

Lemma 1 *Given a sporadic task τ_i, the following inequality holds for all $t \geq d_i$,*

$$\text{RBF}^*(\tau_i, t) \leq \text{DBF}(\tau_i, t) + (u_i \times t) \quad (4)$$

Proof: Observe from the definition of DBF that for $t \geq d_i$, $\text{DBF}(\tau_i, t) \geq e_i$. Substituting this inequality into Equation 3, we obtain Equation 4. ■

3 A Polynomial-Time Partitioning Algorithm

In this paper, we are interested in partitioned feasibility-analysis. Even for implicit-deadline sporadic task systems under the preemptive model, partitioned feasibility-analysis is NP-hard in the strong sense (by transformation from the bin-packing problem [14]). Unfortunately, the complexity results extend to partitioned feasibility-analysis of arbitrary and constrained sporadic task systems.

Bin-packing heuristics have been extensively studied [1, 19]. Typically, each of the bin-packing heuristics adheres to the following pattern:

1. Tasks of the task system are sorted by some criteria.
2. Tasks are assigned (in order) to a processor upon which they "fit" according to a sufficient (and sometimes necessary) condition.

The idea of a task "fitting" on a processor requires some more elaboration: a task τ_i fits on a processor π_k if it can be assigned to the processor, and scheduled according to a given uniprocessor scheduling algorithm such that τ_i and all previously-assigned tasks will always meet all deadlines. The criteria for "fitting" on a processor is thus dependent on the choice of scheduling algorithm at the uniprocessor level. In this paper, we are only considering static-priority scheduling algorithms. Deadline-monotonic scheduling (DM) is known to be optimal for the static-priority scheduling of constrained sporadic task systems on uniprocessors [16]. DM assigns to each task τ_i a priority equivalent to $\frac{1}{d_i}$ and schedules, at any time, the active task with the highest priority. In general, DM performs relatively well in simulations for arbitrary task systems [5]. Therefore, DM is an appropriate algorithm to use to schedule the tasks on each uniprocessor. For the remainder of this paper, we will consider a task to fit on a processor if it can be scheduled according to DM with respect to all tasks previously assigned to the processor.

Section 3.1 presents a partitioning algorithm for sporadic task system where DM is used on each processor. Section 3.2 shows that this algorithm is correct. Section 3.3 shows the partitioning algorithm runs in time polynomial in the number of tasks in the task system.

3.1 Algorithm FBB-FFD

We now describe a simple partitioning algorithm called FBB-FFD. ("FBB" are the letters of the authors' last names, and "FFD" stands for first-fit-decreasing). Given a sporadic task system τ comprised of n sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$, and a processing platform Π comprised of m unit-capacity processors $\pi_1, \pi_2, \ldots, \pi_m$, FBB-FFD will attempt to partition τ among the processors of Π. The FBB-FFD algorithm is a variant of a bin-packing heuristic known as *first-fit-decreasing*. For this section, we will assume the tasks of τ_i are indexed in non-decreasing order of their relative deadline (i.e. $d_i \leq d_{i+1}$, for $1 \leq i < n$).

The FBB-FFD algorithm considers the tasks in decreasing DM-priority order (i.e. $\tau_1, \tau_2, \ldots$). We will now describe how to assign task τ_i assuming that tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ have already successfully been allocated among the m processors. Let $\tau(\pi_\ell)$ denote the set of tasks already assigned to processor π_ℓ where $1 \leq \ell \leq m$. Considering the processors $\pi_1, \pi_2, \ldots, \pi_m$ in any order, we will assign task τ_i to the first processor π_k, $1 \leq k \leq m$, that satisfies the following two conditions:

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

and

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

If no such π_k exists, then Algorithm FBB-FFD returns PARTITIONING FAILED: it is unable to conclude that sporadic task system τ is feasible upon the m-processor platform. Otherwise, FBB-FFD returns PARTITIONING SUCCEEDED.

Constrained Task Systems. We may eliminate the need for checking Inequality 6 by considering constrained task systems. For these systems, it is sufficient to check only Inequality 5:

Lemma 2 *For a constrained sporadic task system τ, any $\tau_i \in \tau$ and $\pi_k \in \Pi$ satisfying Inequality 5, while attempting to assign τ_i to π_k in* FBB-FFD*, will also satisfy Inequality 6.*

Proof: Observe that Inequality 5 implies

$$d_i - \textstyle\sum_{\tau_j \in \tau(\pi_k)} (e_j + d_i \times u_j) \geq e_i$$
$$\Rightarrow \quad 1 - \left(\textstyle\sum_{\tau_j \in \tau(\pi_k)} \frac{e_j}{d_i} \right) - \left(\textstyle\sum_{\tau_j \in \tau(\pi_k)} u_j \right) \geq \frac{e_i}{d_i}.$$

Since $d_i \leq p_i$,

$$1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq \frac{e_i}{p_i} = u_i.$$

Thus, Inequality 6 will evaluate to "true." ■

3.2 Proof of Correctness

In order to prove that FBB-FFD is correct, we are obligated to show that after each assignment of a task to a processor, the system is DM-feasible. In particular, we must prove that if FBB-FFD returned PARTITIONING SUCCEEDED then the set of tasks assigned to each processor is DM-feasible on that processor (Theorem 1). However, before we can prove the correctness of FBB-FFD, we need a feasibility test for uniprocessors that uses RBF*. The following lemma provides such a test:

Lemma 3 *Tasks $\tau(\pi_k)$ are* DM*-feasible on processor π_k if for each $\tau_i \in \tau(\pi_k)$ and $a \in \mathbb{N}^+$ the following condition is satisfied:*

$$\exists t : \; (a-1)p_i < t \leq (a-1)p_i + d_i :: \left(ae_i + \textstyle\sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} \text{RBF}^*(\tau_j, t) \leq t \right) \tag{7}$$

Proof: By Lemma 9 of [11], if Inequality 7 is satisfied, then algorithm APPROX$(\tau(\pi_k), 0.5)$ (described in [11]) will return "$\tau(\pi_k)$ is DM-feasible on π_k." By Theorem 5 of [11], APPROX is correct, and the lemma follows. ■

The next lemma shows that algorithm FBB-FFD maintains the invariant that Inequalities 5 and 6 remain true for all assigned tasks during every step of the algorithm. The invariant is useful to show that the assignment of a task to a processor does not affect the feasibility of the previously-assigned tasks.

Lemma 4 *For each $\pi_k \in \Pi$, the following conditions always hold for algorithm* FBB-FFD*: for every $\tau_j \in \tau(\pi_k)$,*

$$\left(d_j - \sum_{\substack{\tau_\ell \in \tau(\pi_k) \\ \ell < j}} \text{RBF}^*(\tau_\ell, d_j) \right) \geq e_j \tag{8}$$

and

$$1 - \sum_{\substack{\tau_\ell \in \tau(\pi_k) \\ \ell < j}} u_\ell \geq u_j. \tag{9}$$

Proof: Observe that Inequality 8 or 9 for some $\pi_k \in \Pi$ can only be falsified by the assignment of a task τ_i by algorithm FBB-FFD. Thus, we only need to show that Lemma 4 is maintained after every task assignment. We will prove this by induction on the assignment of tasks:

Base Case: Prior to any assignment of a task to a processor by FBB-FFD, each processor is empty. Therefore, Inequalities 8 and 9 are vacuously true.

Inductive Step: Assume Inequalities 8 and 9 remain true for the assignments of $\tau_1, \tau_2, \ldots, \tau_{i-1}$ (by the inductive hypothesis). We must show that Inequalities 8

and 9 continue to hold after the assignment of τ_i. Let τ_i be assigned to processor π_k by FBB-FFD. Observe that Inequalities 8 and 9 for $\pi_s \neq \pi_k$ are unaffected and remain true. Additionally, for $\tau_j \in \tau(\pi_k) - \{\tau_i\}$, it must be that $j < i$, since tasks are assigned in order by FBB-FFD. Thus, for all $\tau_j \in \tau(\pi_k) - \{\tau_i\}$, Inequalities 8 and 9 are unaffected as well. The lemma follows as FBB-FFD ensures that Inequalities 8 and 9 are true (via Inequalities 5 and 6) for τ_i and π_k upon assigning τ_i.

■

Finally, we are prepared to prove the correctness of algorithm FBB-FFD.

Theorem 1 *If* FBB-FFD *returns* PARTITIONING SUCCEEDED*, then the tasks of τ assigned to processors of Π are* DM*-feasible on their respective processors.*

Proof: The proof is by contradiction. Since FBB-FFD returned PARTITIONING SUCCEEDED, then each task of τ is assigned to a processor of Π. For the sake of contradiction, assume there exists a processor $\pi_k \in \Pi$ such that each task $\tau(\pi_k)$ will not always meet all deadlines when scheduled on π_k. By Lemma 3, this implies that there exists a task $\tau_i \in \tau(\pi_k)$ and $a \in \mathbb{N}^+$ such that

$$\forall t : (a-1)p_i < t \leq (a-1)p_i + d_i :: \left(ae_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} \text{RBF}^*(\tau_j, t) > t \right). \quad (10)$$

By Lemma 4, each task-processor assignment satisfies Inequalities 8 and 9. Inequality 8 implies

$$e_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} (e_j + d_i u_j) \leq d_i \text{ (by defititon of RBF}^*). \quad (11)$$

The next equation follows from multiplying both sides of Inequality 9 by $(a-1)p_i$:

$$[(a-1)p_i] \left(u_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} u_j \right) \leq (a-1)p_i. \quad (12)$$

By summing the Inequalities 11 and 12, we obtain

$$ae_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} (e_j + u_j [(a-1)p_i + d_i]) \leq (a-1)p_i + d_i$$

$$\Rightarrow ae_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} \text{RBF}^*(\tau_j, (a-1)p_i + d_i) \leq (a-1)p_i + d_i.$$

The last inequality contradicts Inequality 10. Therefore, our supposition that there exists a π_k where $\tau(\pi_k)$ does not always meet all deadlines is incorrect. It follows that for each $\pi_k \in \Pi$, $\tau(\pi_k)$ is DM-feasible on π_k. ■

3.3 Computational Complexity

Obviously, to sort each task in (non-decreasing) relative deadline order requires $\Theta(n \lg n)$ time. In attempting to map task τ_i, observe that Algorithm FBB-FFD essentially evaluates, in Equations 5 and 6, the workload generated by the previously-mapped $(i-1)$ tasks on each of the m processors. Since $\text{RBF}^*(\tau_j, t)$ can be evaluated in constant time (see Equation 3), a straightforward computation of this workload would require $\mathcal{O}(i+m)$ time. Hence the runtime of the algorithm in mapping all n tasks is no more than $\sum_{i=1}^{n} \mathcal{O}(i+m)$, which is $\mathcal{O}(n^2)$ under the reasonable assumption that $m \leq n$.

4 Theoretical Evaluation

In this section, we quantitatively evaluate the effectiveness of FBB-FFD by providing sufficient conditions for success (Section 4.1) and in terms of a resource augmentation approximation bounds (Section 4.2).

Given a task system τ, the following notation and terminology will be useful for our analysis.

$u_{\max}(\tau)$	$\stackrel{\text{def}}{=} \max_{i=1}^{n}\{u_i\}$	(*max. utilization*)
$u_{\text{sum}}(\tau)$	$\stackrel{\text{def}}{=} \sum_{i=1}^{n} u_i$	(*system utilization*)
$\delta_{\max}(\tau)$	$\stackrel{\text{def}}{=} \max_{i=1}^{n}\{\frac{e_i}{d_i}\}$	(*max. load ratio*)
$\delta_{\text{sum}}(\tau)$	$\stackrel{\text{def}}{=} \max_{t>0}\left(\frac{\sum_{i=1}^{n} \text{DBF}(\tau_i, t)}{t}\right)$	(*system load*)

4.1 Sufficient Schedulability Conditions

The main results of this section (Theorems 2 and 3) provide an upper-bound on the minimum number of processors necessary for FBB-FFD to successfully partition a sporadic task system. The bound is dependent on the utilization and load parameter of the task system.

Before presenting the main theorems of this section, we require several technical lemmas. The next lemma characterizes the conditions under which Inequalities 5 or 6 are trivially satisfied.

Lemma 5 *Given a sporadic task system τ and an m unit-capacity processor system Π,* FBB-FFD *has the following properties:*

P1: *If $u_{\text{sum}}(\tau) \leq 1$, Inequality 6 is always satisfied.*

P2: *If $u_{\text{sum}}(\tau) \leq 1$ and $\delta_{\text{sum}}(\tau) \leq 1 - u_{\text{sum}}(\tau)$, then Inequality 5 is always satisfied.*

Proof: P1 is trivially true, since violating Inequality 6 requires that $(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_k)$ exceed 1.

To see P2, observe that $u_{\text{sum}}(\tau) \leq 1$ and $\delta_{\text{sum}}(\tau) \leq 1 - u_{\text{sum}}(\tau)$ implies for all $t \geq 0$,

$$\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, t)}{t} \leq 1 - u_{\text{sum}}(\tau)$$
$$\Rightarrow \quad \sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) \leq d_i - d_i u_{\text{sum}}(\tau)$$
$$\Rightarrow \quad \sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) + d_i u_{\text{sum}}(\tau) \leq d_i. \qquad (13)$$

Consider any $\tau_i \in \tau$. Observe that for all $j < i$, $d_j \leq d_i$. Thus, for all $j \leq i$, $\text{DBF}(\tau_j, d_i) \geq e_j$. Observing that $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\} \subset \tau$ and combining lower-bound on DBF with Inequality 13 implies

$$e_i + \sum_{j=1}^{i-1} e_j + d_i \left(\sum_{j=1}^{i-1} u_j \right) \leq d_i$$
$$\Rightarrow \qquad e_i + \sum_{j=1}^{i-1} (e_j + d_i u_j) \leq d_i$$
$$\Rightarrow \qquad d_i - \sum_{j=1}^{i-1} \text{RBF}^*(\tau_j, d_i) \geq e_i \qquad (14)$$

P2 follows from Inequality 14 and noting that for all $\pi_k \in \Pi$ $\tau(\pi_k) \subseteq \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$. ■

Corollary 1 *Any sporadic task system τ with $(u_{\text{sum}}(\tau) \leq 1 \wedge \delta_{\text{sum}}(\tau) \leq 1 - u_{\text{sum}}(\tau))$ can be successfully partitioned using* FBB-FFD *on $m \geq 1$ processors.*

The next two lemmas provide an upper-bound on the number of processor on which either Inequality 5 or 6 of FBB-FFD will evaluate to false.

Lemma 6 *Let m_1 denote the number of processors on which Inequality 5 fails while* FBB-FFD *attempts to assign τ_i to a processor. It must be the case that*

$$m_1 < \frac{\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} \qquad (15)$$

Proof: Let $\Pi_1 \subseteq \Pi$ be the set of all processors on which Inequality 5 fails. Then, for all $\pi_k \in \Pi_1$,

$$d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{RBF}^*(\tau_j, d_i) < e_i$$
$$\Rightarrow \quad d_i - \sum_{\tau_j \in \tau(\pi_k)} (\text{DBF}(\tau_j, d_i) + d_i u_j) < e_i. \qquad (16)$$

Inequality 16 follows from Inequality 4 of Lemma 1. By noting that for each π_k, $\tau(\pi_k)$ is a subset of $\tau - \{\tau_i\}$, and summing Inequality 16 over all $\pi_k \in \Pi_1$, we obtain,

$$m_1 d_i - \sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) - d_i u_{\text{sum}}(\tau) < m_1 e_i - e_i$$
$$\Rightarrow \quad m_1(d_i - e_i) + e_i < \sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) + d_i u_{\text{sum}}(\tau)$$
$$\Rightarrow \qquad m_1(1 - \tfrac{e_i}{d_i}) + \tfrac{e_i}{d_i} < \delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau)$$

The last inequality implies the lemma. ■

Lemma 7 *Let m_2 denote the number of processors on which Inequality 6 fails and Inequality 5 is satisfied while* FBB-FFD *attempts to assign τ_i to a processor. It must be the case that*

$$m_2 < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \qquad (17)$$

Proof: Let $\Pi_2 \subseteq \Pi - \Pi_1$ be the set of all processors on which Inequality 6 fails (while Inequality 5 is satisfied). Then, for all $\pi_k \in \Pi_2$,

$$1 - \sum_{\tau_j \in \tau(\pi_k)} u_j < u_i.$$

Noting that for each π_k, $\tau(\pi_k)$ is a subset of τ, and summing Inequality 4.1 over all $\pi_k \in \Pi_1$, we obtain,

$$m_2 - u_{\text{sum}}(\tau) < m_2 u_i - u_i$$
$$\Rightarrow \quad m_2(1 - u_i) < u_{\text{sum}}(\tau) - u_i$$

The last inequality implies the lemma. ■

We are now prepared to prove the sufficient conditions for the success of FBB-FFD over a set of constrained task systems (Theorem 2) and arbitrary task systems (Theorem 3).

Theorem 2 *Any constrained sporadic task system τ is successfully scheduled by* FBB-FFD *on m unit-capacity processors for*

$$m \geq \frac{\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \delta_{\max}(\tau)} \qquad (18)$$

Proof: We will prove the contrapositive of the theorem. Assume that FBB-FFD fails to assign task τ_i to any processor of Π. By Lemma 2, Inequality 5 is false for every $\pi_k \in \Pi$ (i.e. $m = m_1$). Thus, by Lemma 6,

$$m < \frac{\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} \qquad (19)$$

Corollary 1 implies that $u_{\text{sum}}(\tau) > 1$ or $\delta_{\text{sum}}(\tau) > 1 - u_{\text{sum}}(\tau)$. Both inequalities imply that $\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) > 1$. Therefore, the right-hand-side of Inequality 19 is maximized when $\frac{e_i}{d_i}$ is as large as possible. It follows that

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

which proves the contrapositive of the theorem. ■

Theorem 3 *Any sporadic task system τ is successfully scheduled by* FBB-FFD *on m unit-capacity processors for*

$$m \geq \frac{\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \delta_{\max}(\tau)} + \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - u_{\max}(\tau)} \qquad (20)$$

Proof: We will prove the contrapositive of the theorem. Assume that FBB-FFD fails to assign task τ_i to any processor of Π. We now consider four subcases based on the values of $u_{\text{sum}}(\tau)$ and $\delta_{\text{sum}}(\tau)$. Each of the subcases, is either not possible or will imply the contrapositive of the theorem.

1. $\underline{u_{\text{sum}}(\tau) \leq 1 \wedge \delta_{\text{sum}}(\tau) \leq 1 - u_{\text{sum}}(\tau)}$: By Corollary 1, τ would be trivially partitionable on a single processor by FBB-FFD. Therefore, this subcase is impossible as it contradicts our supposition that FBB-FFD fails to assign some task a processor.

2. $\underline{u_{\text{sum}}(\tau) \leq 1 \wedge \delta_{\text{sum}}(\tau) > 1 - u_{\text{sum}}(\tau)}$: By Lemma 5, Inequality 6 is always satisfied. Therefore, Inequality 5 must be violated for all π_k when attempting to assign τ_i (i.e. $m = m_1$). Using reasoning identical to Theorem 2, we will obtain

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

from which the contrapositive of the theorem follows.

3. $\underline{u_{\text{sum}}(\tau) > 1 \wedge \delta_{\text{sum}}(\tau) \leq 1 - u_{\text{sum}}(\tau)}$: Notice that $u_{\text{sum}}(\tau) > 1$ implies that $\delta_{\text{sum}}(\tau) < 0$. This subcase is trivially impossible.

4. $\underline{u_{\text{sum}}(\tau) > 1 \wedge \delta_{\text{sum}}(\tau) > 1 - u_{\text{sum}}(\tau)}$: Recall that m_1 denotes the number of processor on which Inequality 5 of FBB-FFD fails while attempting to assign τ_i. m_2 denotes the remaining processors on which Inequality 6 fails. Therefore, $m = m_1 + m_2$. From Lemma 7,

$$m_2 < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i}. \tag{21}$$

Lemma 6 implies

$$m_1 < \frac{\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}. \tag{22}$$

Summing Inequalities 22 and 21, we obtain

$$m = m_1 + m_2 < \frac{\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} + \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i}. \tag{23}$$

Since $u_{\text{sum}}(\tau) > 1$ and $\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) > 1$, the right-hand-side of Inequality 23 is maximized when both $\frac{e_i}{d_i}$ and u_i are as large as possible. This implies the contrapositive of the theorem.

■

4.2 Resource Augmentation

To better quantify the effectiveness of FBB-FFD in partitioning sporadic task systems, we can use a technique know as *resource augmentation* [20]. Resource augmentation compares a given algorithm against a hypothetical optimal algorithm and determines the factor by which we augment the processing platform for the given algorithm to match the performance of the optimal. For the purpose of this paper, the resource augmentation technique is as follows: given a task system that is known to be feasible (global or partition) upon a particular platform, we determine the multiplicative factor of speed by which we must augment our platform in order for FBB-FFD to return PARTITIONING SUCCEEDED (shown in Theorem 4).

First, we need a technical lemma that characterizes the necessary conditions for feasibility. In [7], Baruah and Fisher showed for sporadic task systems that the larger of $\delta_{\max}(\tau)$ and $u_{\max}(\tau)$ represents the maximum computational demand of any task, and the larger of $\delta_{\text{sum}}(\tau)$ and $u_{\text{sum}}(\tau)$ represents the maximum computational demand of a sporadic task system τ.

Lemma 8 (from [7]) *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*

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

and

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

Theorem 4 *Given an identical multiprocessor platform Π with m processors and a sporadic task system τ (global or partition) feasible on Π , the* FBB-FFD *algorithm has the following performance guarantees:*

1. *if τ is a $\underline{\text{constrained}}$ system, then* FBB-FFD *will successfully partition τ upon a platform comprised of m processors that are each $\left(3 - \frac{1}{m}\right)$ times as fast as the processors of Π.*

2. *if τ is an $\underline{\text{arbitrary}}$ system, then* FBB-FFD *will successfully partition τ upon a platform comprised of m processors that are each $\left(4 - \frac{2}{m}\right)$ times as fast as the processors of Π.*

Proof: Assume that we are given task system τ feasible on m processors each of speed ξ, it follows from Lemma 8 that τ must satisfy the following properties:

$$\begin{array}{ll} \delta_{\text{sum}}(\tau) \leq m \cdot \xi & u_{\text{sum}}(\tau) \leq m \cdot \xi \\ \delta_{\max}(\tau) \leq \xi & u_{\max}(\tau) \leq \xi \end{array} \tag{24}$$

Suppose that τ is successfully scheduled on m unit-capacity processor by FBB-FFD.

We first show (1) by substituting the Inequalities of 24 above into Inequality 18 of Theorem 2:

$$\begin{array}{rcl} m \geq \frac{\delta_{\text{sum}}(\tau) + u_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \delta_{\max}(\tau)} & & \\ \Leftarrow & m \geq \frac{m\xi + m\xi - \xi}{1 - \xi} & \\ \Leftarrow & m \geq \frac{2m\xi - \xi}{1 - \xi} & \\ \equiv & \xi \leq \frac{m}{3m - 1} & \\ \equiv & \frac{1}{\xi} \geq 3 - \frac{1}{m}. & \end{array}$$

The last implication is claimed by (1) of the theorem.

To show (2), we similarly substitute the Inequalities of 24 into Inequality 20 of Theorem 3:

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

which is claimed by (2) of the theorem. ■

4.3 Comparison with Prior Implicit-Deadline Partitioning Results

Polynomial-time partitioning algorithms for implicit-deadline systems are known [19, 1]. To evaluate the theoretical loss of schedulability resulting from the move to a more general task model, let us consider the resource augmentation bound of the partitioning algorithm analyzed by Andersson and Jonsson [1]. By using the logic of Theorem 4, the following theorem can be obtained from their $0.5m$ utilization bound:

Theorem 5 *Given an identical multiprocessor platform* Π *with* m *processors and a implicit-deadline sporadic task system* τ *(global or partition) feasible on* Π*, can be partitioned in polynomial-time upon a platform comprised of* m *processors that are each approximately 2 times as fast as the processors of* Π*.*

5 Simulation

In addition to the theoretical evaluation of the previous section, we have evaluated FBB-FFD by simulating its execution over a set of randomly generated task systems. We compare FBB-FFD with an algorithm RT-FFD that uses a necessary and sufficient condition for a task fitting on a processor (see [12] for details of RT-FFD which is based on determining the worst-case response time of each task). The goal of this section is to characterize the schedulability loss of FBB-FFD due to using a sufficient (but not necessary) polynomial-time test for feasibility on each processor. Section 5.1 briefly describes our simulation evaluation methodology. Section 5.2 presents the results of the simulation of both algorithms over the test sets.

5.1 Methodology

We evaluated FBB-FFD and RT-FFD over several datasets. Each dataset contains 1,000,000 task systems with at most 63 tasks per system. Each task's period was pseudo-randomly generated from the uniform distribution of integers between 1 and 1000. Below are the different parameters we used to generate each dataset:

1. Utilization Distribution: We used four different distributions to pseudo-randomly generate parameter u_i for task τ_i (note e_i can immediately be determined from p_i and u_i):
   a) Uniform Distribution: between $\frac{1}{p_i}$ and 1.0.
   b) Bimodal Distribution: generate *heavy* and *light* tasks; probability of task being heavy is $1/3$; utilization for heavy tasks is drawn uniformly from 0.5 to 1.0; utilization for light is drawn uniformly from $\frac{1}{p_i}$ to 0.5 , excluding 0.5.
   c) Exponential Distribution with mean of 0.25.
   d) Exponential Distribution with mean of 0.5.
2. Relative Deadline Parameter: We considered three different distributions for pseudo-randomly generating d_i parameter:
   a) Constrained: d_i uniformly distributed between e_i and p_i.
   b) Super-period: $d_i = kp_i$ where k is uniformly drawn from the set $\{1, 2, 3, 4\}$.
   c) Unconstrained (tri-modal distribution): tasks can have pre-period deadline ($d_i < p_i$), post-period deadline ($d_i > p_i$), or implicit deadline ($d_i = p_i$) with uniform probability; pre-period deadlines are drawn uniformly from e_i to p_i; post-period deadlines are generated in an identical manner to super-period distributions.
3. Number of Processors (m): 2, 4, and 8 processors systems were considered.

We have tested FBB-FFD and RT-FFD over all 36 combinations. However, due to space we will only report representative combinations in the next subsection. The other combinations not reported display similar trends.

Tasks are generated in the following manner: $m + 1$ initial tasks are generated and tested by running FBB-FFD and RT-FFD. Then another task is pseudo-randomly generated and added to the previous set. and FBB-FFD and RT-FFD are run over the resulting task set. The process is iterated until the resulting task load ($\delta_{\text{sum}}(\tau)$) exceeds m (see [6] for details on calculating $\delta_{\text{sum}}(\tau)$). Once the load exceeds m, a new initial set of $m+1$ tasks is generated and the procedure is repeated.

5.2 Simulation Results

We will only show the results for datasets generated for four processors. The datasets for two and eight processors exhibited similar trends for RT-FFD and FBB-FFD. Each of the plots (Figures 1–5) displays a histogram for the

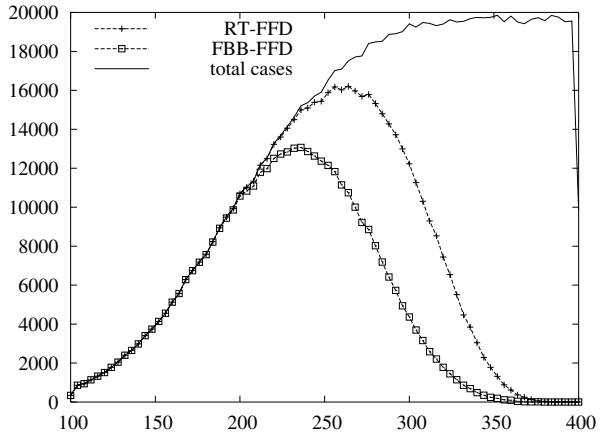

Figure 1. Constrained deadline, bimodal distribution, 4 CPUs. See Section 5.2 for description of the axes for each plot.

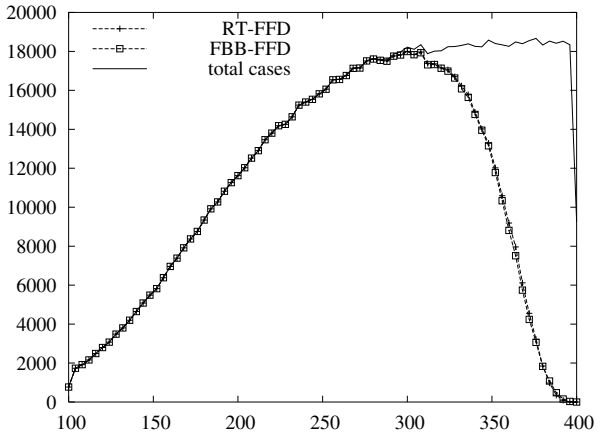

Figure 2. Super-period deadline, bimodal distribution, 4 CPUs

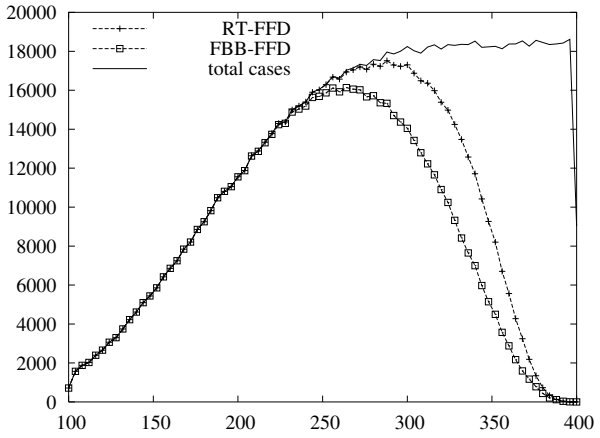

Figure 3. Unconstrained deadline, bimodal distribution, 4 CPUs

number of task sets schedulable by FBB-FFD and RT-FFD with respect to load. The x-axis represents the value of $\delta_{\text{sum}}(\tau) * 100$, and the y-axis represents the number of task systems schedulable according to each algorithm. In addition, the solid line in each plot represents a histogram of the total number of task sets generated with respected to load. Systems with $\delta_{\text{sum}}(\tau) < 1$ are trivially feasible and are not shown in the plots

Figures 1, 2, and 3 give the results for the bimodal distribution. The schedulability loss for FBB-FFD (when compared to RT-FFD) is largest for constrained deadlines (Figure 1) and negligible for super-period deadlines (Figure 2 – both algorithms can partition the same number of task systems). The reason for the difference in schedulability loss is due decreased probability of any task having a large individual task load (e_i/d_i) for super-period deadlines. Figure 3 displays the results for unconstrained when the task systems contain both pre- and post-period deadline tasks. Figure 4 and 5 shows the effect of the deadline selection rule for the exponential distribution with a mean utilization of 0.25.

Even though RT-FFD can schedule a greater or nearly equal number of task systems than FBB-FFD for all distributions we have tested, we have observed that RT-FFD and FBB-FFD are incomparable in general. That is there exist tasks that can be partitioned by FBB-FFD but not RT-FFD, and vice versa. The incomparability of the two algorithms is not surprising as they are both heuristics for an NP-hard problem.

6 Conclusions

The development of general models of real-time computation has contributed to the breadth and maturity of uniprocessor scheduling research. Our goal is to broaden the applicability and generality of multiprocessor systems by studying the merits of general task models for multiprocessor scheduling. This paper focuses on the partitioned scheduling of general sporadic task systems where a job's deadline can differ from the task's period. For such systems, we develop a polynomial-time partitioning algorithm on multiprocessor platforms. In addition, we quantitatively evaluate our algorithm by deriving sufficient conditions for success and obtaining resource augmentation approximation bounds. To obtain polynomial-time complexity, our algorithm use a sufficient condition for determining whether to assign a task to a processor. We empirically evaluate the schedulability loss incurred from using only a sufficient feasibility condition by comparing the performance with exact conditions for fitting on a processor. We hope to apply the techniques from this paper to obtaining feasibility and schedulibility conditions for uniform multiprocessors and even more general models of real-time computation.

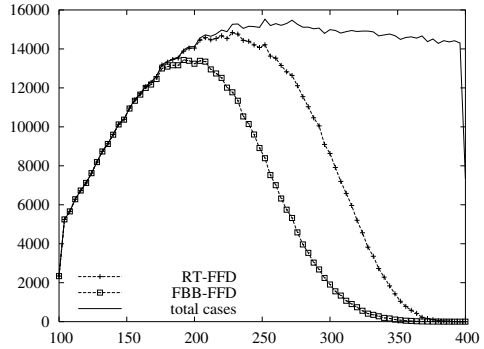

Figure 4. Unconstrained deadline, exponential w/mean utilization of 0.25, 4 CPUs

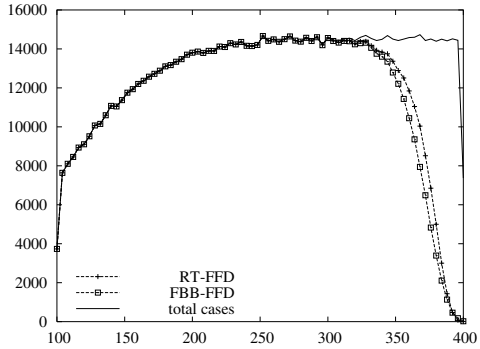

Figure 5. Super-period deadline, exponential w/mean utilization of 0.25, 4 CPUs

References

[1] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Porto, Portugal, July 2003. IEEE Computer Society Press. Refer to Errata for corrected proofs.

[2] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 127–132, Atlanta, May 1991.

[3] T. P. Baker. An analysis of deadline-monotonic schedulability on a multiprocessor. Technical Report TR-030201, Department of Computer Science, Florida State University, 2003.

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

[5] 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, 2005.

[6] T. P. Baker, N. Fisher, and S. Baruah. Algorithms for determining the load of a sporadic task system. Technical Report TR-051201, Department of Computer Science, Florida State University, 2005.

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

[8] 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.

[9] 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 2005. IEEE Computer Society Press.

[10] 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 2005. IEEE Computer Society Press.

[11] N. Fisher and S. Baruah. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 117–126, Palma de Mallorca, Balearic Islands, Spain, July 2005. IEEE Computer Society Press.

[12] N. Fisher and S. Baruah. The partitioned, static-priority scheduling of sporadic real-time tasks with constrained deadlines on multiprocessor platforms. In *Proceedings of the 9th International Conference on Principles of Distributed Systems*, Pisa, Italy, December 2005. IEEE Computer Society Press.

[13] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. In *Proceedings of the 13th International Conference on Real-Time Systems*, Paris, France, April 2005.

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

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

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

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

[18] 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.

[19] D.-I. Oh and T. P. Baker. Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems: The International Journal of Time-Critical Computing*, 15:183–192, 1998.

[20] 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.