The Partitioned Multiprocessor Scheduling of Non-preemptive Sporadic Task Systems*

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

Abstract

We consider polynomial-time algorithms for partitioning a collection of non-preemptive or restricted-preemption tasks among the processors of an identical multiprocessor platform. Since the problem of partitioning tasks among processors (even with unlimited preemption) is NP-hard in the strong sense, these algorithms are unlikely to be optimal. For task systems where the ratio between the largest execution time and the smallest relative deadline is small, we provide a sufficient condition for feasibility. The application of this algorithm to preemptive quantum-based systems is also discussed. For all other task systems, we experimentally evaluate different variants of our heuristic over sets of randomly generated tasks.

1 Introduction

In many real-time systems, complete *a priori* knowledge of job release times is either impractical or impossible. The **sporadic task model** [16] provides a characterization of real-time computation of such task systems by allowing time between the release of successive jobs of a task to vary. For a sporadic task τ_i, a *minimum inter-arrival separation* parameter (historically called the *period*) describes the minimum time interval between successive jobs of a task. A collection of jobs generated by the sporadic task system is called *legal* if the minimum inter-arrival separation is respected for each task. A *relative deadline* parameter identifies the time interval from a job's release time to its absolute deadline during which execution of the job must complete. A collection of sporadic tasks is called a *sporadic task system.*

The advantage of preemptive scheduling on uniprocessors has been known since Lui and Layland [14] showed that total utilization of a uniprocessor is achievable under preemptive EDF scheduling. In preemptive scheduling, a job may be halted before the completion of its execution and resumed at a later time. However, in many systems, preemption is impractical or undesirable due to the high overhead involved in context switching between different tasks. In *non-preemptive scheduling*, once a job begins execution it executes continuously on the processor until its completion. Non-preemptive scheduling for these systems can reduce scheduling overhead, and have the following additional benefits [12]: elimination of the need for complex resource sharing protocols for resources or critical sections that are local to a processor; reduction in the implementation complexity of scheduling protocols; and estimates of worst-case execution time for tasks may be more accurate in the non-preemptive model.

In addition to pure non-preemptive tasks, it is sometimes useful to model both preemptive and non-preemptive behavior in the same system. The *restricted-preemption model* allows tasks to execute non-preemptively in short intervals, and be preempted in-between the non-preemptive intervals. Each task specifies a *non-preemption parameter* which indicates the maximum length of time a task may non-preemptively execute. Notice that a non-preemptive system can be represented in the restricted preemption model by setting the non-preemption parameter for each task equal to its execution requirement. A preemptive system can be represented by setting the non-preemption parameter for each task to zero. The restricted-preemption model is also useful for characterizing the behavior of *quantum-based* scheduling systems.

An important endeavor in real-time scheduling theory is determining whether a task system is *feasible* on a given processing platform. A task system is feasible if there exists, for each legal collection of job releases, a schedule on the processing platform in which no task misses a deadline. For non-preemptive and restricted-preemption systems, a more restrictive notion of feasibility can be defined: *feasibility without inserted idle times (IIT)*. A task system is feasible without IIT if there exists a schedule for every legal collection of jobs in which a processor is never idle while there are jobs awaiting execution. A scheduling algorithm is *optimal* in this model if it can schedule every task system that is feasible without IIT.

Uniprocessor Scheduling. The non-preemptive real-time scheduling of sporadic tasks has been studied extensively for uniprocessor platforms. For sporadic task systems where each task's relative deadline is equal to its period, Jeffay et al. [12] proved that the non-preemptive *Earliest Deadline First* algorithm (EDF) [14] is optimal on uniprocessors with respect to scheduling without IIT, and

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

they provided necessary and sufficient conditions for EDF-schedulability. Non-preemptive EDF schedules at each idle instant the job with the nearest deadline (from the set of jobs awaiting execution). George et al. removed the restriction on a task's relative deadline, showed that EDF is optimal without IIT [10], and provided modified necessary and sufficient conditions for EDF-schedulability [11]. Researchers have also focused different techniques and models [2, 9, 6] for limiting preemptions in an attempt to obtain the benefits of both preemptive and non-preemptive scheduling.

Multiprocessor Scheduling. For the multiprocessor scheduling of non-preemptive and restricted-preemption sporadic tasks, two alternative paradigms exist: *global* and *partitioned* scheduling. For restricted-preemption and non-preemptive global scheduling, a job executing a non-preemptive code section will execute continuously on the same processor; a job executing a preemptive code-section can be halted and can resume execution on a different processor. For non-preemptive partitioned scheduling, a task is assigned to a processor, and all jobs of the task are always executed on that processor.

Non-preemptive and restricted-preemption multiprocessor scheduling of sporadic tasks has received much less attention. Baruah [5] considered the non-preemptive global scheduling of periodic and sporadic tasks on an identical multiprocessor platform. To the best of our knowledge, there has been no work done on the partitioned scheduling of non-preemptive sporadic task systems.

This research. We consider the non-preemptive and restricted-preemption multiprocessor scheduling of sporadic task systems under the partitioned paradigm. Since partitioning tasks among processors reduces the multiprocessor scheduling algorithm to a series of uniprocessor scheduling problems (one to each processor), the optimality (without IIT) of non-preemptive EDF [12, 10] makes EDF a reasonable algorithm to use as the run-time scheduler on each processor. Therefore, we henceforth make the assumption that each processor, and the tasks assigned to it by the partitioning algorithm, are scheduled during runtime according to non-preemptive EDF and focus on the partitioning algorithm.

Recently, Albers and Slomka [1] developed a fully polynomial-time approximation scheme for determining the feasibility of preemptive sporadic task systems on a uniprocessor. Their results also extend to non-preemptive sporadic task systems. In [7], we non-trivially applied the results of Albers and Slomka to obtain a polynomial-time partitioning algorithm for preemptive sporadic task systems. In this paper, we extend and generalize the results in [7] to be applicable for non-preemptive and restricted-preemption systems. To address some of the drawbacks of the simple restricted-preemption partitioning algorithm that we derive, we also consider a family of heuristics which we evaluate experimentally.

Organization. In Section 2, we present background on the non-preemptive and restricted-preemption sporadic task model. We also discuss uniprocessor feasibility tests for restricted-preemption sporadic tasks. In Section 3, we design a simple algorithm for partitioning restricted-preemption systems, and evaluate this algorithm theoretically. The application of this algorithm to preemptive quantum-based systems is explored as well. In Section 4, we consider some more pragmatic heuristics for partitioning and evaluate their performance empirically. In Section 5, we draw together the conclusions of this paper.

2 Task and machine model

A *restricted-preemption sporadic task* [6] $\tau_i = (e_i, q_i, d_i, p_i,)$ is characterized by a *worst-case execution requirement* e_i, a *non-preemption parameter* q_i, a *(relative) deadline* d_i, and a *minimum inter-arrival separation* p_i. In general, a restricted-preemption sporadic task is subject to the trivial constraints that $q_i \leq e_i$, $e_i \leq p_i$, 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 $u_i \stackrel{\text{def}}{=} e_i/p_i$. The non-preemption parameter q_i specifies the maximum length of time at which τ_i may execute non-preemptively on a single processor. Observe that this parameter specifies only the interval length during which τ_i executes non-preemptively, not the start and end times of the non-preemptive interval. In our model, τ_i may execute non-preemptively for up to q_i time units starting at any point in time, and may do so arbitrarily often (subject to the constraints of the other task parameters). We may model a pure non-preemptive sporadic task by setting $q_i = e_i$.

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 restricted-preemption sporadic tasks where $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and $\tau_i = (e_i, d_i, p_i, q_i)$ 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.

2.1 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 $t_o + p_i$, $t_o + 2p_i$, $t_o + 3p_i, \ldots$ Equation (1) below follows directly [8]:

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

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

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

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 $t \geq 0$:

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

2.2 Uniprocessor Feasibility

It has been shown [12] that the response time of a task τ_i (i.e. the time from a job's release to its completion) is maximized in the following scenario: Let τ_k $(k > i)$ be the task with the largest non-preemptive execution requirement. The worst-case sequence is if a job of τ_k is released just prior to the release of a job of τ_i, and all other tasks τ_j $(j \neq i, k)$ release jobs simultaneously with τ_i; in addition, successive jobs of τ_j are released as soon as legally possible. Any feasibility test for non-preemptive sporadic systems must determine whether a deadline miss will occur in this scenario. The feasibility test we use for non-preemptive and restricted-preemptions uniprocessor systems is from [6]:

Theorem 1 (from [6]) *A restricted-preemption sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ is feasible (without IIT) if and only if*

$$\forall t : 0 \leq t : \sum_{i=1}^{n} \text{DBF}(\tau_i, t) \leq t, \quad (4)$$

and for all $\tau_j = (e_j, q_j, d_j, p_j)$ where $1 \leq j \leq n$

$$\forall t : 0 \leq t \leq d_j : q_j + \sum_{\substack{i=1 \\ i \neq j}}^{n} \text{DBF}(\tau_i, t) \leq t \quad (5)$$

Throughout the remainder of this paper, we will adopt the convention that "feasibility" is with respect to restricted-preemption model without IIT.

3 A Partitioning Algorithm

Given a system of sporadic tasks, the problem of determining whether it is possible for the task system to always satisfy all timing constraints is called *feasibility-analysis*. 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 [13]). Unfortunately, the complexity results extend to partitioned feasibility-analysis of all restricted-preemption (or non-preemptive) sporadic task systems.

For the preemptive model, several bin-packing heuristics have been studied [15]. 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.

In Section 3.1, we describe such a partitioning algorithm. Section 3.2 derives sufficient conditions for schedulability of a restricted-preemption task system using this algorithm. The algorithm and the derivation of the sufficient conditions generalizes work done for partitioning of preemptive sporadic task systems in [7]. Section 3.3 describes an application of this algorithm for preemptive quantum-based scheduling.

3.1 Algorithm NP-PARTITION

We now describe a simple partitioning algorithm called NP-PARTITION. Given a restricted-preemption 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 $\pi_1, \pi_2, \ldots, \pi_m$, NP-PARTITION will attempt to partition τ among the processors of Π. The NP-PARTITION 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 $1 \leq i < n$). Let $q_{\max}(\tau) \stackrel{\text{def}}{=} max_{i=1}^{n}\{q_i\}$ denote the maximum non-preemption parameter of any task in τ.

The NP-PARTITION algorithm considers the tasks in increasing index 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:

$$\left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) \geq e_i + q_{\max}(\tau) \quad (6)$$

and

$$\left(1 - \sum_{\tau_j \in \tau(\pi_k)} u_j\right) \geq u_i \; ; \tag{7}$$

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

If τ is a constrained, sporadic task system, then it suffices to check only Equation 6:

Lemma 1 *For* **constrained** *sporadic task systems, any τ_i and π_k satisfying Equation 6 during execution of the* NP-PARTITION *algorithm will also satisfy Equation 7*

Proof: Observe that for any constrained task τ_i, Equation 2 implies that for all $t \geq d_i$,

$$\text{DBF}^*(\tau_i, t) = u_i \times (t + p_i - d_i) \geq u_i \times t.$$

Hence, Equation 6

$$\begin{array}{rcl} \equiv & \left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \geq e_i + q_{\max}(\tau)\right) \\ \Rightarrow & d_i - \sum_{\tau_j \in \tau(\pi_k)} (u_j \times d_i) \geq e_i + q_{\max}(\tau) \\ \Rightarrow & 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq \frac{e_i}{d_i} + \frac{q_{\max}(\tau)}{d_i} \\ \Rightarrow & 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq u_i + \frac{q_{\max}(\tau)}{d_i} \end{array} \tag{8}$$

The last inequality implies Equation 7. ■

We must now show that by assigning task τ_i to processor π_k we have not adversely affected the feasibility of tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ previously assigned to the processors. The next lemma shows that the system remains EDF-feasible if we assign tasks according to Equations 6 and 7.

Lemma 2 *If the tasks previously assigned to each processor were* EDF*-feasible (with respect to restricted-preemption) on that processor and Algorithm* NP-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 assigning τ_i to processor π_k does not affect the tasks previously assigned to other processors. Therefore, we focus our attention only on π_k, and show that if $\tau(\pi_k)$ was EDF-feasible prior to the addition of τ_i, and Equation 6 and 7 are satisfied, then $\tau(\pi_k) \cup \{\tau_i\}$ remains EDF-feasible after the addition of τ_i.

For the sake of contradiction, assume that $\tau(\pi_k)$ and τ_i satisfies Equation 6 and 7 of NP-PARTITION, but that EDF misses a deadline when scheduling the tasks $\tau(\pi_k) \cup \{\tau_i\}$ on processor π_k. Let t_f be the time that processor π_k misses a deadline. Observe that $t_f > d_i$ since $\tau(\pi_k)$ is EDF-feasible before the addition of τ_i.

By Theorem 1, either

$$\text{DBF}(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}(\tau_j, t_f) > t_f \tag{9}$$

or there exists $\tau_\ell \in \tau(\pi_k) \cup \{\tau_i\}$ such that

$$q_\ell + \sum_{\substack{\tau_j \in \tau(\pi_k) \cup \{\tau_i\} \\ j \neq \ell}} \text{DBF}(\tau_j, t_f) > t_f. \tag{10}$$

We will show that if either Equation 9 or 10 is true, then a contradiction is reached. Assume that Equation 9 is true. Then, since DBF* is an upper bound on DBF,

$$\text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) > t_f \tag{11}$$

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

$$\begin{array}{rcll} \text{DBF}^*(\tau_j, t_f) & = & e_j + u_j(t_f - d_j) & \text{(By definition)} \\ & = & e_j + u_j(d_i - d_j) + u_j(t_f - d_i) & \\ & = & \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i) & (12) \end{array}$$

Furthermore,

$$\begin{array}{rcl} & & \text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) \\ & \equiv & (e_i + u_i(t_f - d_i)) + \quad \textit{(By Equation 12 above)} \\ & & \left(\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i)\right) \\ & \equiv & \left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) \\ & & + (t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) \end{array}$$

Consequently, Inequality 11 above can be rewritten as follows:

$$\left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) + (t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) > (t_f - d_i) + d_i \tag{13}$$

However by Condition 6, $(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)) \leq d_i - q_{\max}(\tau) \leq d_i)$; Inequality 13 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 7.

Now, assume that Equation 10 is true. Similar to Equation 11, we obtain

$$q_{\max}(\tau) + \sum_{\substack{\tau_j \in \tau(\pi_k) \cup \{\tau_i\} \\ j \neq \ell}} \text{DBF}^*(\tau_j, t_f) > t_f. \qquad (14)$$

Following similar logical steps of Equations 12 through 13, we would obtain from Equation 14:

$$(u_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \cup \{\tau_i\} \\ j \neq \ell}} u_j) > 1.$$

which also contradicts Equation 7.

Since both Equations 9 and 10 lead to a contradiction, our supposition that processor π_k missed a deadline at time t_f is false. Thus, $\tau(\pi_k) \cup \{\tau_i\}$ will always meet all deadlines on processor π_k. ■

The correctness of Algorithm NP-PARTITION follows, by repeated applications of Lemma 2:

Theorem 2 *If Algorithm* NP-PARTITION *returns* PARTITIONING SUCCEEDED *on task system τ, then the resulting partitioning is* EDF*-feasible.*

Run-time complexity. In attempting to map task τ_i, observe that Algorithm NP-PARTITION essentially evaluates, in Equations 6 and 7, 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 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 $m \leq n$.

3.2 Theoretical Evaluation

In this section, we derive a set of sufficient conditions for the success of NP-PARTITION. In particular, for each subclass of restricted-preemption sporadic tasks we derive a different sufficient condition: Theorem 3 corresponds to implicit-deadline systems, Theorem 4 for constrained systems, and Theorem 5 for arbitrary systems.

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

$d_{\min}(\tau)$	$\stackrel{\text{def}}{=} \min_{i=1}^{n}\{d_i\}$	
$\rho(\tau)$	$\stackrel{\text{def}}{=} q_{\max}(\tau)/d_{\min}(\tau)$	(*max. blocking ratio*)
$u_{\max}(\tau)$	$\stackrel{\text{def}}{=} \max_{i=1}^{n}\{u_i\}$	(*max. utilization*)
$u_{\text{sum}}(\tau)$	$\stackrel{\text{def}}{=} \sum_{i=1}^{n} u_i$	(*system utilization*)
δ_i	$\stackrel{\text{def}}{=} e_i/d_i$	(*task load ratio*)
$\delta_{\max}(\tau)$	$\stackrel{\text{def}}{=} \max_{i=1}^{n}\{\delta_i\}$	(*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*)

The following lemma describes subcases where either Equation 6 or 7 is trivially satisfied. The corollary immediately following shows that certain combination of subcases imply that τ is trivially restricted-preemption feasible on a single processor. The lemma and corollary will be useful in proving the main results of Theorems 3, 4, and 5.

Lemma 3 *Given a restricted-preemption sporadic task system τ and an m unit-capacity processor system Π,* NP-PARTITION *has the following properties:*

P1: *If $u_{\text{sum}}(\tau) \leq 1$, Equation 7 is always satisfied.*

P2: *If $\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2}$, then Equation 6 is always satisfied.*

P3: *Let τ be an implicit-deadline system. If $u_{\text{sum}}(\tau) \leq 1 - \rho(\tau)$, then both Equations 6 and 7 are always satisfied.*

Proof: P1 is trivially true, since violating Equation 7 requires that $(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j)$ exceed 1.

To see P2, observe that $\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2}$ implies that $\sum_{\tau_j \in \tau} \text{DBF}(\tau_j, t_0) \leq \frac{t_0(1-\rho(\tau))}{2}$ for all $t_0 \geq 0$. By Inequality 3, this in turn implies that

$$\sum_{\tau_j \in \tau} \text{DBF}^*(\tau_j, t_0) \leq t_0(1-\rho(\tau)) \qquad (15)$$

for all $t_0 \geq 0$; specifically at $t_0 = d_i$ when evaluating Equation 6 for τ_i. Evaluating Equation 15 at $t_0 = d_i$ implies that $e_i + \sum_{j=1}^{i-1} \text{DBF}^*(\tau_j, d_i) \leq d_i - q_{\max}(\tau)$. Since $\tau(\pi_k) \subseteq \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ when attempting to add τ_i to processor π_k in NP-PARTITION, $e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \leq d_i - q_{\max}(\tau)$. This implies Equation 6.

To see P3, observe that

$$u_{\text{sum}}(\tau) \leq 1 - \rho(\tau) \qquad (16)$$

trivially implies Equation 7. It remains to show that Equation 6 is satisfied. In an implicit-deadline system, it follows from Equation 2 that DBF^* can be rewritten as:

$$\text{DBF}^*(\tau_i, t) = \begin{cases} 0, & \text{if } t < d_i \\ u_i t, & \text{otherwise} \end{cases} \qquad (17)$$

By multiplying both sides of Equation 16 by d_i, we obtain

$$\begin{aligned} & \textstyle\sum_{j=1}^{n} u_j d_i \leq d_i - q_{\max}(\tau) \\ \Rightarrow \quad & e_i + \textstyle\sum_{\substack{j=1 \\ j \neq i}}^{n} u_j d_i \leq d_i - q_{\max}(\tau) \\ \Rightarrow \quad & e_i + \textstyle\sum_{\substack{j=1 \\ j \neq i}}^{n} \text{DBF}^*(\tau_j, d_i) \leq d_i - q_{\max}(\tau) \\ \Rightarrow \quad & e_i + \textstyle\sum_{\substack{j=1 \\ j \neq i}}^{i-1} \text{DBF}^*(\tau_j, d_i) \leq d_i - q_{\max}(\tau). \end{aligned}$$

Since $\tau(\pi_k) \subseteq \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ when attempting to add τ_i to processor π_k in NP-PARTITION, the last inequality implies Equation 6. ■

Corollary 1

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

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

3. *Any implicit-deadline, restricted-preemption sporadic task system τ satisfying $(u_{\text{sum}}(\tau) \leq 1-\rho(\tau))$ is successfully partitioned on any number of processors ≥ 1 by* NP-PARTITION.

Proof: Part 1 and 3 follow directly from Lemma 3, Equations 6 and 7 of NP-PARTITION will always evaluate to "true."

Part 2 follows directly from Lemmas 3 and 1. By Lemma 1, we need only determine that Equation 6 is satisfied. By Property P2 of Lemma 3, this is ensured by having $\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2}$.

■

We are now prepared to prove sufficient conditions for NP-PARTITION successfully partitioning an implicit-deadline, restricted-preemption sporadic task system.

Theorem 3 *Any implicit-deadline, restricted-preemption sporadic task system τ where $\rho(\tau) < 1 - u_{\text{sum}}(\tau)$ is successfully scheduled by* NP-PARTITION *on m unit-capacity processors, for any*

$$m \geq \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - \rho(\tau) - u_{\max}(\tau)} \tag{18}$$

Proof Sketch: The proof is by contradiction. Assume that Equation 18 is true, but NP-PARTITION fails to partition τ. Then there exists a task $\tau_i \in \tau$ such that when attempting to add τ_i to each processor $\pi_k \in \Pi$, either Equation 6 or 7 is violated. Corollary 1 implies for each $\pi_k \in \Pi$

$$u_{\text{sum}}(\tau(\pi_k) \cup \{\tau_i\}) > 1 - \rho(\tau) \tag{19}$$

because otherwise when attempting to assign τ_i to processor π_k, NP-PARTITION would be able to "fit" $\tau(\pi_k)$ and τ_i on the same processor.

Summing Inequality 19 over all $\pi_k \in \Pi$, and noting that the tasks on these processors is a subset of τ, we obtain

$$\begin{aligned} & u_{\text{sum}}(\tau) + (m-1)u_i > m(1-\rho(\tau)) \\ \Rightarrow \quad & m(1-\rho(\tau)-u_i) < u_{\text{sum}}(\tau) - u_i \\ \Rightarrow \quad & m < \tfrac{u_{\text{sum}}(\tau)-u_i}{1-\rho(\tau)-u_i} \end{aligned}$$

Observe that $u_{\text{sum}}(\tau) > 1 - \rho(\tau)$; otherwise, τ would be trivially feasible according to Corollary 1. Therefore, the left-hand side of the above inequality is maximized when u_i is as large as possible. This implies,

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

which contradicts our assumption, thereby proving the lemma. ■

In the next lemmas, we describe the necessary conditions for algorithm NP-PARTITION failing to assign τ_i to a processor in Π. If τ_i was not assigned to a processor then either Equation 6 or 7 evaluated to false for each $\pi_k \in \Pi$. Lemma 4 quantifies the maximum number of processors for which Equation 6 is false; Lemma 5 quantifies the maximum number of processors for which Equation 7 is false.

Lemma 4 *Let m_1 denote the number of processors, $0 \leq m_1 \leq m$, on which Equation 6 fails when the partitioning algorithm is attempting to map τ_i. Assuming that $\rho(\tau) < 1 - \delta_i$,it must be the case that*

$$m_1 < \frac{2\delta_{\text{sum}}(\tau) - \delta_i}{1 - \rho(\tau) - \delta_i} \tag{20}$$

Proof: Let Π_1 be the set of m_1 processor for which Equation 6 evaluates to false when attempting to add task τ_i. Then, for each $\pi_k \in \Pi_1$,

$$\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) > d_i - e_i - q_{\max}(\tau)$$

Summing over all $\pi_k \in \Pi_1$, and noting that $\bigcup_{\pi_k \in \Pi_1} \tau(\pi_k) \subseteq \tau$, we obtain

$$\begin{aligned} & \sum_{j=1}^{n} \text{DBF}^*(\tau_j, d_i) > (d_i - e_i - q_{\max}(\tau))m_1 + e_i \\ \Rightarrow \quad & \text{(by Inequality 3)} \\ & 2\textstyle\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) > (d_i - e_i - q_{\max}(\tau))m_1 + e_i \\ \Rightarrow \quad & \tfrac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} > \tfrac{m_1}{2}(1 - \delta_i - \tfrac{q_{\max}(\tau)}{d_i}) + \tfrac{e_i}{2d_i} \end{aligned} \tag{21}$$

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

$$\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} \leq \delta_{\text{sum}}(\tau). \tag{22}$$

Chaining Inequalities 21 and 22, and observing that $\rho(\tau) \geq \frac{q_{\max}(\tau)}{d_i}$, we obtain

$$\begin{aligned} & \tfrac{m_1}{2}(1 - \delta_i - \rho(\tau)) + \tfrac{e_i}{2d_i} < \delta_{\text{sum}}(\tau) \\ \Rightarrow \quad & m_1 < \tfrac{2\delta_{\text{sum}}(\tau) - \delta_i}{1 - \rho(\tau) - \delta_i} \end{aligned}$$

which is claimed by the lemma. ■

Lemma 5 *Let m_2 denote the number of processors, $0 \leq m_2 \leq m$, on which Equation 7 fails when the partitioning algorithm is attempting to map τ_i. It must be the case that*

$$m_2 < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \tag{23}$$

Proof: Let Π_2 be the set of m_2 processor for which Equation 7 evaluates to false when attempting to add task τ_i. Then, for each $\pi_k \in \Pi_2$,

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

Summing over all $\pi_k \in \Pi_2$, and noting that $\bigcup_{\pi_k \in \Pi_2} \tau(\pi_k) \subseteq \tau$, we obtain

$$\begin{aligned} & (1-u_i)m_2 + u_i < \sum_{j=1}^{n} u_j \\ \Rightarrow \quad & m_2 < \tfrac{u_{\text{sum}}(\tau)-u_i}{1-u_i} \end{aligned}$$

which is asserted by the lemma. ■

We are now prepared to prove sufficient conditions for success of NP-PARTITION on constrained and arbitrary task systems.

Theorem 4 *Any constrained, restricted-preemption sporadic task system τ where $\rho(\tau) < 1 - \delta_{\max}(\tau)$ is successfully scheduled by* NP-PARTITION *on m unit-capacity processors, for any*

$$m \geq \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \rho(\tau) - \delta_{\max}(\tau)} \tag{24}$$

Proof: The proof is by contradiction. Assume that task system τ and platform Π satisfy Inequality 24, but that NP-PARTITION fails to assign some task $\tau_i \in \tau$ to any processor. By Lemma 1, it must be the case that Equation 6 fails for task τ_i on each of the m processors (i.e. m equals m_1 from Lemma 4). Consequently, Lemma 4 implies

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

By the second part of Corollary 1, $\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2}$; otherwise, τ can trivially be partitioned by NP-PARTITION. In this case, the right-hand side of the above inequality is maximized when δ_i is as large as possible. Thus,

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

which contradicts Inequality 24. ■

Theorem 5 *Any restricted-preemption sporadic task system τ where $\rho(\tau) < 1 - \delta_{\max}(\tau)$ is successfully scheduled by* NP-PARTITION *on m unit-capacity processors, for any*

$$m \geq \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \rho(\tau) - \delta_{\max}(\tau)} + \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - u_{\max}(\tau)} \tag{25}$$

Proof Sketch: The proof is by contradiction. Assume that task system τ and platform Π satisfy Inequality 25, but that NP-PARTITION fails to assign some task $\tau_i \in \tau$ to any processor. Let Π_1 be the set of m_1 processor on which Equation 6 evaluates to false while attempting to assign τ_i. Let Π_2 be the remaining m_2 processors (i.e. $m_2 \stackrel{\text{def}}{=} m - m_1$) on which Equation 7 evaluates to false.

According to Part 1 of Corollary 1, $(u_{\text{sum}}(\tau) > 1)$ or $(\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2})$; otherwise, NP-PARTITION could trivially partition τ on a single processor. We will consider three separate cases and show that in each case a contradiction will arise. Due to space requirements we only fully show the first case.

Case(i): ($\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2}$ and $u_{\text{sum}}(\tau) > 1$).: In this case, both m_1 and m_2 are non-zero. Summing Inequalities 20 and 23 of Lemmas 4 and 5 (respectively), we obtain

$$m = m_1 + m_2 < \frac{2\delta_{\text{sum}}(\tau) - \delta_i}{1 - \rho(\tau) - \delta_i} + \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i}$$

Because ($\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2}$), ($u_{\text{sum}}(\tau) > 1$), and ($\rho(\tau) < 1 - \delta_{\max}(\tau)$), the right-hand side of the above inequality is maximized when both δ_i and u_i are as large as possible. Therefore,

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

which contradicts Inequality 25.

Case(ii): ($\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2}$ and $u_{\text{sum}}(\tau) \leq 1$). Similar to Case (i) via a simple application of Lemma 4.

Case(iii): ($\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2}$ and $u_{\text{sum}}(\tau) > 1$). Similar to Case (i) via a simple application of Lemma 5.

■

3.3 Quantum-based Preemptive Scheduling

A *quantum-based* scheduler allocates a processor to tasks in "blocks" of time called *quantums*. During a quantum a task may execute non-preemptively until the earliest of the following two events occurs: the task completes execution or its quantum expires. In the event of task completion, the scheduler is invoked, and the next available task (according to the scheduling algorithm) with remaining execution is assigned the newly idle processor. If the quantum expires prior to the completion of the task, the task is preempted and the scheduling algorithm makes a decision about what task to execute in the next quantum (possibly the same task). In this model, a job of a task can be "blocked" by a lower-priority (assuming a priority-based scheduling algorithm) only if it arrives in the middle of a quantum. Therefore, the maximum blocking time for any task in the system is equal to one quantum. We will make the following simplifying assumptions about the system:

1. The scheduler is invoked only at the completion of the execution of a job, or at the end of the quantum.
2. The scheduler's execution requirement is negligible.
3. The quantum-size is less than the execution requirement of each task in the system.
4. The quantum sizes are fixed and identical on each processor.

NP-PARTITION is a highly applicable partitioning algorithm for these quantum-based systems. We can model a quantum-based system by setting, for each task $\tau_i \in \tau$, q_i equal to the quantum-size. Since NP-PARTITION reserves $q_{\max}(\tau)$ units of "slack" at every instance of time on each processor, this guarantees that each task assigned to a processor can still meet its deadline despite being blocked for a quantum's duration of time. In fact, even an optimal partitioning algorithm for a multiprocessor quantum-based must ensure that a job can be delayed

for one quantum. Therefore, we can evaluate the performance of NP-PARTITION in quantum-based systems using a technique know as *resource augmentation* [17]. Resource algorithm 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 upon a particular platform (with respect to quantum-based scheduling), we determine the multiplicative factor of speed by which we must augment our platform in order for NP-PARTITION to return PARTITIONING SUCCEEDED (shown in Theorem 6).

First, we need a technical lemma that characterizes the necessary conditions for feasibility. In [7], Baruah and Fisher showed for preemptive 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 preemptive sporadic task system τ. This result trivially extends to restricted-preemption systems.

Lemma 6 (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 6 *Given an identical multiprocessor platform Π with m processors and a restricted-preemption sporadic task system τ feasible on Π without IIT, the* NP-PARTITION *algorithm has the following performance guarantees for a quantum-based system with quantum-size equal to $q_{\max}$ (i.e. $\rho(\tau) = \frac{q_{\max}}{d_{\min}(\tau)}$):*

1. *if τ is an implicit-deadline system, then* NP-PARTITION *will successfully partition τ upon a platform comprised of m processors that are each $\left(\frac{2-\frac{1}{m}}{1-\rho(\tau)}\right)$ times as fast as the processors of Π.*
2. *if τ is a constrained system, then* NP-PARTITION *will successfully partition τ upon a platform comprised of m processors that are each $\left(\frac{3-\frac{1}{m}}{1-\rho(\tau)}\right)$ times as fast as the processors of Π.*
3. *if τ is an arbitrary system, then* NP-PARTITION *will successfully partition τ upon a platform comprised of m processors that are each $\left(\frac{4-\frac{2}{m}}{1-\rho(\tau)}\right)$ times as fast as the processors of Π.*

Proof Sketch: Due to space considerations, we will only show the proof of the third claim. The other claims can be proven similarly.

Assume that we are given arbitrary task system τ feasible (without IIT) on m processors each of speed ξ, it follows from Lemma 6 that τ must satisfy the following properties:

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

Suppose that τ is successfully scheduled on m unit-capacity processor by NP-PARTITION. By substituting the inequalities above in Equation 25 of Theorem 5, we get

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

which is claimed in the third part of the theorem. ■

4 Heuristics

Though NP-PARTITION is useful for theoretically evaluating preemptive quantum-based systems, it pessimistically assumes that any task in the system could potentially blocked for $q_{\max}(\tau)$ time units (Equation 6). Therefore, the algorithm is impractical for all but small values of $\rho(\tau)$ and entirely unusable for $\rho(\tau) > 1$. In a general restricted-preemption system, we may not need assume the same maximum non-preemption parameter each processor. In this section, we will consider a family of polynomial-time heuristics based on the first-fit-decreasing bin-packing heuristic. Section 4.1 will describe each of the heuristics we consider. Section 4.2 will present our empirical-evaluation methodology and results. We will also discuss potential theoretic justifications for the experimental results.

4.1 Heuristic Descriptions

As mentioned in the beginning of Section 3, typical partitioning heuristics will first sort the tasks according to some criteria, and then assign the tasks to a processor according to a sufficient condition for "fitting". The NP-PARTITION algorithm matches this heuristic pattern by sorting tasks in (non-decreasing) order of relative deadline and adding tasks to processors (in order) according to Equations 6 and 7. In this section, we consider a family of algorithms that each sorts tasks in non-increasing order according to different criteria and assigns a task to the first processor upon which it fits (each algorithm is a variant of first-fit-decreasing). The conditions for fitting are the same for each heuristic considered and are a more optimistic sufficient conditions than used for NP-PARTITION (i.e. a task is more likely to be assigned to a processor). The following are the eleven different sorting criteria: $\frac{1}{d_i}$ (same as NP-PARTITION), $\frac{1}{p_i}$, e_i, q_i, u_i, $\lambda_i \stackrel{\text{def}}{=} \frac{e_i}{\min(d_i,p_i)}$,

δ_i, $\widehat{u}_i \stackrel{\text{def}}{=} \frac{q_i}{p_i}$, $\widehat{\lambda}_i \stackrel{\text{def}}{=} \frac{q_i}{\min(d_i,p_i)}$, $\widehat{\delta}_i \stackrel{\text{def}}{=} \frac{q_i}{d_i}$, and *random* order. Each heuristic is denoted by FFD-NP-$\langle$*sorting-criteria*$\rangle$.

For each the above heuristics, we will attempt to add tasks $\tau_i \in \tau$ to processors of Π in the order specified. Since we have potentially changed the order that tasks are assigned, Lemma 2 is not necessarily valid. Therefore, every time we assign a task τ_i to a processor π_k, we must check that each task in $\tau(\pi_k)$ meets its deadline after the addition of τ_i. We will add a task τ_i to processor π_k only if it does not affect the EDF-feasibility of the tasks of $\tau(\pi_k)$.

Equations 6 and 7 of the NP-PARTITION algorithm ensured that Equations 4 and 5 of Theorem 1 are not violated, by leaving enough slack on a processor to fit the job with the largest non-preemption parameter. We can consider a more optimistic test by making the following observation: *since we are considering* EDF*-schedulability, a task τ_j on a processor may only be blocked (i.e. a priority inversion occurs) if a task with a larger relative deadline is executing when a job of τ_j arrives.* Therefore, when checking if task τ_i fits on a processor π_k, we need only check if all tasks $\tau_j \in \tau(\pi_k)$ with $d_j < d_i$ have enough slack to accommodate being blocked by τ_i for q_i time units. We must also ensure that the added demand placed on π_k by τ_i after time d_i does not affect the slack necessary to accommodate jobs of $\tau_j \in \tau(\pi_k)$ with $d_j > d_i$. Therefore, we replace Equation 6 with the following conditions: for each $\tau_\ell \in \{\tau\} \cup \tau(\pi_k)$,

$$\left(d_\ell - \sum_{\substack{\tau_j \in \tau(\pi_k) \\ d_j < d_\ell}} \text{DBF}^*(\tau_j, d_\ell) \right) \geq e_\ell + \max_{\substack{\tau_j \in \{\tau\} \cup \tau(\pi_k) \\ d_j > d_\ell}} \{q_j\}. \quad (26)$$

It turns out that if Equations 26 and 7 are true, then we can safely add τ_i to processor π_k. The full proof of correctness is omitted due to space considerations.

Run-time Complexity. For each $\pi_k \in \Pi$, let i_k be the number of tasks assigned to processor π_k at the time we are attempting to assign task τ_i. For each $\tau_\ell \in \{\tau_i\} \cup \tau(\pi_k)$, it takes $\mathcal{O}(i_k)$ time to evaluate the $\sum \text{DBF}^*(\tau_j, d_\ell)$ term and $\mathcal{O}(1)$ time to evaluate $\max\{q_j\}$ term in Equation 26. Therefore, the time complexity for testing if τ_i fits on processor π_k is $\mathcal{O}(i_k^2)$. This implies the overall time-complexity of each of the heuristics is $\mathcal{O}(n^4)$.

4.2 Empirical Evaluation

Methodology. For empirically evaluating each of the heuristics, we generated a set of pseudo-random tasks sets in a manner similar to Baker [3]. We pseudo-randomly generate a partition of $m+1$ preemptive tasks. That is, each processor is assigned a set of randomly generated tasks which are preemptively feasible on the processor, and the total number of tasks in this initial set is $m+1$. The method of generating an individual task $\tau_i = (e_i, q_i, d_i, p_i)$ is as follows: we generate a random utilization parameter u_i for the task according to the exponential distribution with mean individual task utilization of 0.25. We then randomly generate the period p_i from a uniform distribution with values ranging from 1 to 1000. The execution requirement e_i is computed from the task utilization and period. The relative deadline d_i of a task is an integer value chosen from the uniform distribution with range $[e_i, 2p_i]$.

A partition of these $m+1$ tasks is randomly generated such that each task is preemptively feasible. If $\tau(\pi_k)$ is the set of tasks from the initial task set assigned to processor π_k, the condition of feasibility is $\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t) \leq t$ for all $t \geq 0$. After determining an initial partition, we compute a maximum non-preemption parameter $\hat{q}_i$ for each task in this partition, and assign to q_i a value chosen from the exponential distribution with range $[0, \min(e_i, \hat{q}_i)]$. To generate subsequent task sets, we add a randomly generated task to the current partition and attempt to fit it on any processor. If the task fits on the processor, we add the resulting task set to our sample; otherwise, we start over with a newly generated set of $m+1$ tasks.

We generated several different sets of task systems according to the methodology described. However, due to space consideration, we discuss only one set of 1,000,000 randomly generated task systems. Each of the task systems generated are feasible on a four processor platform. Figure 1 shows the distribution of generated task systems with respect to a task system's δ_{sum} value ([4] gives justification for using δ_{sum} as metric of comparison).

Results and Discussion. The results of running each of the eleven heuristics over the sample set are shown in Figure 2. The values are shown for $\delta_{\text{sum}} \geq 2.5$ only, as the smaller δ_{sum}-valued task systems are partitionable by all heuristics. The major trend we have observed from this experiment and other experiments (omitted for space) are that heuristics that have the non-preemption parameter q_i in the sorting criteria (i.e. FFD-NP-q_i, FFD-NP-$\widehat{u}_i$, FFD-NP-$\widehat{\lambda}_i$, and FFD-NP-$\widehat{\delta}_i$) dominate heuristics that exclude q_i for $\delta_{\text{sum}} \leq 3.8$. Indeed, FFD-NP-$q_i$ dominates all heuristics for $\delta_{\text{sum}} \leq 3.8$. A potential reason for the domination of the heuristics using q_i is that by assigning tasks with larger non-preemption parameter first, we more effectively utilize the slack on each processor. If we ignored the q_i parameter in ordering tasks, we may not leave enough room for future tasks with large q_i. However, according to Figure 2, heuristics based on the task load or density (FFD-NP-λ_i and FFD-NP-δ_i) may be more effective than q_i-based heuristics for task systems with high δ_{sum} values. Not surprisingly, FF-NP-RANDOM does the worst and is dominated by all other heuristics since it does not exploit any information about the task system.

5 Conclusions

Non-preemptive scheduling of tasks has the advantage of decreasing system complexity and scheduling overheads. However, little work has been done on studying non-preemption for partitioned multiprocessor systems. In this paper, we considered the parititoned multiprocessor scheduling of restricted-preemption and non-preemptive sporadic task systems. We introduced a simple partitioning algorithm NP-PARTITION for which we

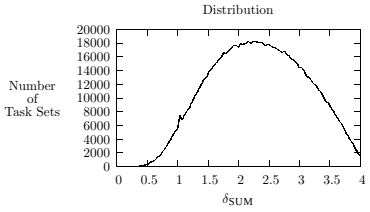

Figure 1. Distribution of δ_{sum} values for randomly generated task sets

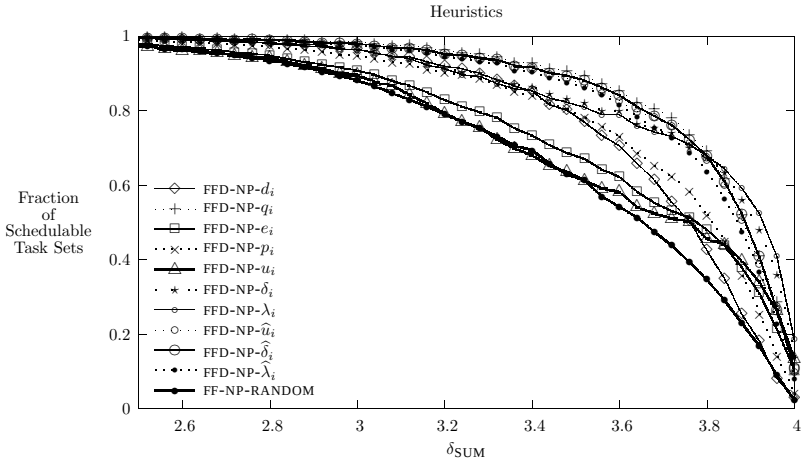

Figure 2. Comparison of partitioning heuristics

derived sufficient conditions for success (Theorems 3, 4, and 5). NP-PARTITION can be applied to partitioning preemptive sporadic tasks in a multiprocessor system that uses quantum-based scheduling. For quantum-based systems, we were able to characterize the effectiveness of NP-PARTITION in terms of resource augmentation bounds (Theorem 6).

To address the drawbacks of the pessimistic behavior of NP-PARTITION, we considered eleven different partitioning heuristics for restricted-preemption sporadic task systems. We characterized the performance of these heuristics empirically over a set of randomly generated task systems, and observed that heuristics which use the non-preemption or load information of the task system outperform heuristics that ignore this information. In the future, we would like to obtain better theoretical bounds for the heuristics presented in Section 4.

References

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

[2] T. P. Baker. Stack-based scheduling of real-time processes. *Real-Time Systems: The International Journal of Time-Critical Computing*, 3, 1991.

[3] 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.

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

[5] S. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems: The International Journal of Time-Critical Computing*. Accepted for publication.

[6] S. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 137–144, Palma de Mallorca, Balearic Islands, Spain, July 2005. IEEE Computer Society Press.

[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] P. Gai, G. Lipari, and M. di Natale. Minimizing memory utilization of real-time task sets in single and multiprocessor systems-on-a-chip. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, December 2001.

[10] L. George, P. Muhlethaler, and N. Rivierre. Optimality and non-preemptive real-time scheduling revisited. Technical Report RR-2516, INRIA: Institut National de Recherche en Informatique et en Automatique, 1995.

[11] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report RR-2966, INRIA: Institut National de Recherche en Informatique et en Automatique, 1996.

[12] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12th Real-Time Systems Symposium*, pages 129–139, San Antonio, Texas, December 1991. IEEE Computer Society Press.

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

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

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

[16] 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.

[17] 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.