The Partitioned Scheduling of Sporadic Real-Time Tasks on Multiprocessor Platforms

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

Abstract— **In the sporadic task model, a task is characterized by three parameters — an execution requirement, a relative deadline, and a period parameter — and has the interpretation that it generates an infinite sequence of jobs, such that (i) the arrival-times of any two successive jobs are separated by a time-interval at least as long as the period parameter; (ii) each job has a deadline that is separated from its arrival-time by a time-interval exactly equal to the relative deadline parameter of the task; and (iii) each job must execute for an amount equal to its execution requirement by its deadline.**

Most previous research concerning the scheduling of collections of sporadic tasks upon multiprocessor platforms has added the additional constraint that all tasks have their relative deadline parameters equal to their period parameters. In this research, we consider the scheduling of systems of sporadic tasks that do not necessarily satisfy this additional constraint, upon preemptive multiprocessor platforms. We propose, and evaluate, an algorithm for partitioning a given collection of arbitrary sporadic tasks upon a specified number of preemptive processors such that all deadlines are guaranteed to always be met.

Index Terms— **Sporadic tasks; partitioned scheduling; shared-memory multiprocessors.**

I. Introduction

Over the years, the sporadic task model [11], [2] has proven remarkably useful for the modelling of recurring processes that occur in hard-real-time systems. In this model, a *sporadic task* $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* e_i, a *(relative) deadline* d_i, and a *minimum inter-arrival separation* p_i, which is, for historical reasons, also referred to as the *period* of the task. (The ratio $u_i \overset{\text{def}}{=} e_i/p_i$ of sporadic task τ_i is often referred to as the *utilization* of τ_i.) Such a sporadic task generates a potentially infinite sequence of jobs, with successive job-arrivals separated by at least p_i time units. Each job has a worst-case execution requirement equal to e_i and a deadline that occurs d_i time units after its arrival time. A *sporadic task system* is comprised of several such sporadic tasks. Let τ denote a system of such sporadic tasks: $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$, with $\tau_i = (e_i, d_i, p_i)$ for all i, $1 \leq i \leq n$.

A system of sporadic tasks is said to be *feasible* upon a specified platform if it is possible to schedule the system within the constraints imposed by 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 systems of sporadic tasks on preemptive *uni*processors has been extensively studied. It is known (see, e.g. [2]) that a uniprocessor system of preemptive sporadic tasks is feasible if and only if all deadlines can be met when each task in the system has a job arrive at the same time-instant, and subsequent jobs arrive as rapidly as legal (such a combination of job-arrival sequences is sometimes referred to as a *synchronous arrival sequence* for the sporadic task system.) This fact, in conjunction with the optimality of the Earliest Deadline First scheduling algorithm (EDF) for scheduling preemptive uniprocessor systems [7], [3], has allowed for the design of preemptive uniprocessor feasibility-analysis algorithms for sporadic task systems [11], [2].

§**Multiprocessor systems.** On multiprocessor systems, two alternative paradigms for scheduling collections of sporadic tasks have been considered: *partitioned* and *global* scheduling. In the partitioned approach, the tasks are statically partitioned among the processors, i.e., each task is assigned to a processor and is always executed on it. Under global scheduling, it is permitted that a job that has previously been preempted from one processor resume execution at a later point in time upon a different processor, at no additional cost (however each job may be executing on at most one processor at each instant in time).

Most prior research on multiprocessor scheduling of collections of sporadic tasks has assumed that *all tasks have their deadlines equal to their period parameters* (i.e., $d_i = p_i$ for all tasks τ_i) — such sporadic systems are sometimes referred to in the literature as

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

implicit-deadline systems. For implicit-deadline systems, feasibility-analysis under the global paradigm is trivial [4], [12]: an implicit-deadline system τ is feasible upon a platform comprised of m unit-capacity processors if and only if **(i)** $u_i \leq 1$ for each task $\tau_i \in \tau$; and **(ii)** $\sum_{\tau_i \in \tau} u_i \leq m$. Under the partitioned paradigm, feasibility-analysis for implicit-deadline systems can be transformed to a bin-packing problem [6] and shown to be NP-hard in the strong sense; sufficient feasibility tests for various bin-packing heuristics have recently been obtained [10], [9].

§**This research.** Our objective in this research is to study the preemptive multiprocessor scheduling of arbitrary sporadic real-time systems under the partitioned paradigm. Since this system model is a generalization of the implicit-deadline model, the intractability result (feasibility analysis being NP-hard in the strong sense) continues to hold; to our knowledge, there are no prior non-trivial positive theoretical results concerning this problem [1]. Our major contribution (Theorem 1) is *a polynomial-time test that is sufficient, although not necessary, for ensuring that a given sporadic task system is feasible upon a specified number of processors*. Our approach towards designing this test is constructive: we devise and analyze, in Section III, a polynomial-time algorithm for partitioning a given sporadic task system among a specified number of processors. This partitioning algorithm attempts to find a mapping from the given collection of tasks to the available processors, such that all the tasks mapped on to a specific processor are guaranteed to be feasible on that particular processor. Since EDF is known to be optimal for scheduling preemptive uniprocessor systems [7], [3], the tasks mapped on to each processor can subsequently be scheduled using EDF in order to guarantee that no deadlines are missed during run-time.

§**Organization.** The remainder of this paper is organized as follows. In Section II, we formally specify the task model used in the remainder of this paper, and define the *demand bound function* as a characterization of sporadic tasks by the *maximum workload* such a task is able to generate over a time interval of given length. In Section III, we present our partitioning algorithm, and formally establish its correctness. In Section IV, we derive some further properties of our algorithm; these properties further characterize its behavior on typical task systems. We illustrate the workings of the algorithm on a simple, 10-task system in Section V. In Section VI we propose an improvement to the algorithm: although this improvement does not seem to effect the worst-case behavior of the algorithm, it is shown to be of use in successfully partitioning some sporadic task systems that our basic algorithm – the one presented in Section III – fails to handle.

[1]"Trivial" results include the obvious ones that τ is feasible on m processors if *(i)* it is feasible on a single processor; or *(ii)* the system obtained by replacing each task τ_i by a task $\tau_i' = (e_i, \min(d_i, p_i), \min(d_i, p_i))$ is deemed feasible using the heuristics presented in [10], [9].

II. Task and machine model

A *sporadic task* $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* e_i, a *(relative) deadline* d_i, and a *minimum inter-arrival separation* p_i. The ratio $u_i \overset{\text{def}}{=} e_i/p_i$ of sporadic task τ_i is often referred to as the *utilization* of τ_i.

We will assume that we have a multiprocessor platform comprised of m identical processors $\pi_1, \pi_2, \ldots, \pi_m$, each of unit computing capacity, on which we are to schedule a system τ of n sporadic tasks: $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$, with $\tau_i = (e_i, d_i, p_i)$ for all i, $1 \leq i \leq n$. Without loss of generality, assume that tasks are indexed according to non-decreasing order of their relative deadline parameter (i.e., $d_i \leq d_{i+1}$ for all i, $1 \leq i < n$).

The 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 [2] 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 [2]:

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

§ **An approximation of** $\text{DBF}(\tau_i, t)$**.** The function $\text{DBF}(\tau_i, t)$, if plotted as a function of t for a given task τ_i, is represented by a series of "steps," each of height e_i, at time-instants d_i, $d_i + p_i$, $d_i + 2p_i$, $\cdots$, $d_i + kp_i$, $\cdots$. Albers and Slomka [1] have proposed a technique for *approximating* the DBF, which tracks the DBF exactly through the first several steps and then approximates it

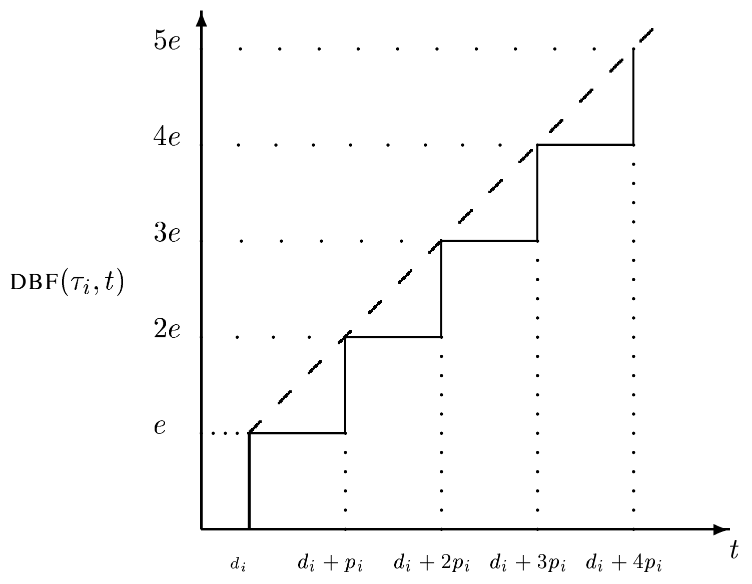

Fig. 1. **The step function denotes a plot of $\text{DBF}(\tau_i, t)$ as a function of t. The dashed line represents the function $\text{DBF}^*(\tau_i, t)$, approximating $\text{DBF}(\tau_i, t)$; for $t < d_i$, $\text{DBF}^*(\tau_i, t) \equiv \text{DBF}(\tau_i, t) = 0$.**

by a line of slope e_i/p_i (see Figure 1). In the following, we are applying this technique in essentially tracking DBF exactly for a *single* step of height e_i at time-instant d_i, followed by a line of slope e_i/p_i.

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

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) satisfies this job-arrival sequence by requiring that the first job's deadline be met explicitly by being assigned e_i units of execution between its arrival-time and its deadline, and that τ_i be assigned $u_i \times \Delta\, t$ of execution over time-interval $[t, t + \Delta\, t)$, for all instants t after the deadline of the first job, and for arbitrarily small positive $\Delta\, t$ (see Figure 2 for a pictorial depiction).

Observe that the following inequality holds for all τ_i and for all $t \geq 0$:

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

with the ratio $\text{DBF}^*(\tau_i, t)/\text{DBF}(\tau_i, t)$ being maximized just prior to the deadline of the second job of τ_i — i.e., at $t = d_i + p_i - \epsilon$ for ϵ an arbitrarily small positive number — in the synchronous arrival sequence; at this time-instant, $\text{DBF}^*(\tau_i, t) \to 2e$ while $\text{DBF}(\tau_i, t) = e$.

§**Comparison of DBF^* and the "density" approximation.** The quantity $[e_i/\min(d_i, p_i)]$ is sometimes referred to as the **density** [8] of sporadic task τ_i. It is known that a sufficient condition for a sporadic task system to be feasible upon a unit-capacity *uni*processor is that the sum of the densities of all tasks in the system not exceed one (see, e.g., [8, Theorem 6.2]). This condition is obtained by essentially approximating the demand bound function $\text{DBF}(\tau_i, t)$ of τ_i by the quantity $(t \cdot \max(u_i, \frac{e_i}{d_i}))$. This approximation is never superior to our approximation DBF^*, and is inferior if $d_i \neq p_i$: for our example in Figure 1, this approximation would be represented by a line with the same slope as the dashed line passing through the origin.

III. A PARTITIONING ALGORITHM

Recall from Section II that we are assuming that tasks are indexed according to non-decreasing order of their relative deadline parameter (i.e., $d_i \leq d_{i+1}$ for all i, $1 \leq i < n$). Our partitioning algorithm considers the tasks in the order $\tau_1, \tau_2, \ldots$. Suppose that tasks τ_1, τ_2, ..., τ_{i-1} have all been successfully allocated among the m processors, and we are now attempting to allocate task τ_i to a processor. Our algorithm for doing this is a variant of the *First Fit* [5] algorithm for bin-packing, and is as follows. For any processor π_ℓ, let $\tau(\pi_\ell)$ denote the tasks from among $\tau_1, \ldots, \tau_{i-1}$ that have already been allocated to processor π_ℓ. Considering the processors in the order $\pi_1, \pi_2, \ldots, \pi_m$, we will assign task τ_i to the first processor π_k, $1 \leq k \leq m$, that satisfies the following two conditions:

$$\left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \right) \geq e_i \quad (4)$$

and

$$\left(1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \right) \geq u_i \ ; \quad (5)$$

If no such π_k exists, then we declare failure: we are unable to conclude that sporadic task system τ is feasible upon the m-processor platform.

The following lemma asserts that, in assigning a task τ_i to a processor π_k, our partitioning algorithm does not adversely affect the feasibility of the tasks assigned to each processor. The correctness of the partitioning algorithm follows, by n applications of this lemma.

Lemma 1: If the tasks previously assigned to each processor were feasible on that processor and the algorithm above assigns task τ_i to processor π_k, then the

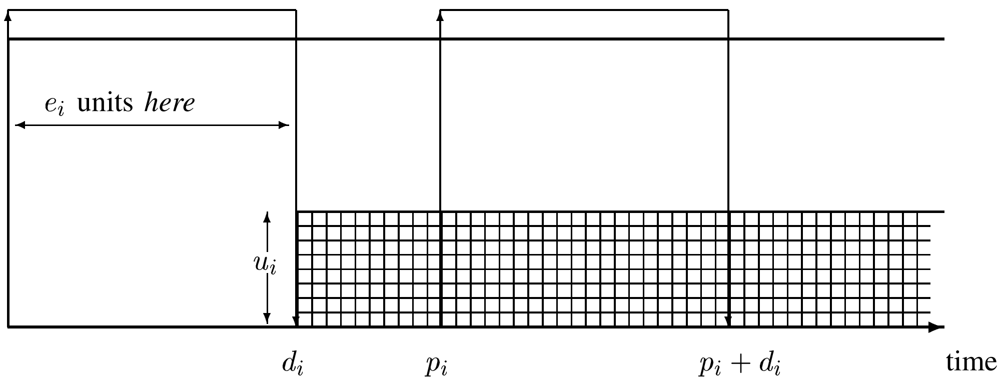

Fig. 2. **Pictorial representation of task τ_i's reservation of computing capacity in a processor-sharing schedule.**

tasks assigned to each processor (including processor π_j) remain feasible on that processor.

Proof: Observe that the feasibility of the processors other than processor π_k is not affected by the assignment of task τ_i to processor π_k. It remains to demonstrate that, if the tasks assigned to π_k were feasible on π_k prior to the assignment of τ_i and Conditions 4 and 5 are satisfied, then the tasks on π_k remain feasible after adding τ_i.

Now the scheduling of processor π_k after the assignment of task τ_i to it is a uniprocessor scheduling problem. It is known (see, e.g. [2]) that a uniprocessor system of preemptive sporadic task is feasible if and only all deadlines can be met for the synchronous arrival sequence (i.e., when each task has a job arrive at the same time-instant, and subsequent jobs arrive as rapidly as legal). Hence to demonstrate that π_k remains feasible after adding task τ_i to it, it suffices to demonstrate that all deadlines can be met for the critical arrival sequence.

To see that all deadlines are indeed met for the critical arrival sequence, observe that our algorithm essentially assigns task τ_i to processor π_k only if there is enough remaining computing capacity on the processor to **(i)** meet τ_i's first deadline (at time-instant d_i) – this is checked by Condition 4; and **(ii)** allocate a fraction u_i of the processor's computing capacity at all time-instants over the interval (d_i, ∞) — this is checked by Condition 5. (See Figure 2 for a visual depiction of this assignment.) Thus, τ_i's first deadline is met by construction. In order to ensure that the ℓ'th deadline is met for all $\ell > 1$, observe that this ℓ'th deadline will occur at time-instant $(d_i + (\ell - 1)p_i)$. Over the interval $(d_i, d_i + (\ell - 1)p_i)$, task τ_i is allocated an amount of execution equal to $(\ell - 1)p_i \cdot u_i$, which is equal to $(\ell - 1) \cdot e_i$ — hence, the ℓ'th deadline is also met. ■

Run-time complexity

In attempting to map task τ_i, observe that our partitioning algorithm essentially evaluates, in Equations 4 and 5, 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 $m \leq n$.

IV. Evaluation

As stated in Section I, our partitioning algorithm represents a sufficient, rather than exact, test for feasibility — it is possible that there are systems that are feasible under the partitioned paradigm but which will be incorrectly flagged as "infeasible" by our partitioning algorithm. Indeed, this is to be expected since a simpler problem – partitioning collections of sporadic tasks that all have their deadline parameters equal to their period parameters – is known to be NP-hard in the strong sense while our algorithm runs in $\mathcal{O}(n^2)$ time. In this section, we offer a quantitative evaluation of the efficacy of our algorithm when various assumptions may be made about the task system.

First, observe that there are two points in our partitioning algorithm during which errors may be introduced. First, we are approximating a solution to a generalization of the bin-packing problem by an *any-fit* heuristic [5] – simply place a task on any processor upon which it fits. Second, we are approximating the demand bound function DBF by the function DBF^*, thereby introducing an approximation factor of at most two (Inequality 3). The first of these sources of errors arises even in the consideration of implicit-deadline systems; however, the second source of error is unique to the generalization in the task model.

Let us now suppose that our partitioning algorithm fails to obtain a partition for sporadic task system τ on m processors. In particular, let us suppose that task τ_i cannot be mapped on to any processor and let Π_1 denote the m_1 processors upon which this mapping fails because Condition 4 is not satisfied (hence for the remaining $m_2 \stackrel{\text{def}}{=} (m - m_1)$ processors, denoted Π_2, Condition 4 is satisfied. but Condition 5 is not). Let us extend previous notation as follows: for any collection of processors Π_x, let $\tau(\Pi_x)$ denote the tasks from among $\tau_1, \ldots, \tau_{i-1}$ that have already been allocated to some processor in the collection Π_x. Lemmas 2 and 3 provide upper bounds on the values of m_1 and m_2, in terms of the parameters of the tasks $\tau_1, \ldots, \tau_i$:

Lemma 2:

$$m_1 < \left(\sum_{\tau_j \in \tau(\Pi_1)} \frac{\text{DBF}^*(\tau_j, d_i)}{d_i - e_i} \right) \tag{6}$$

Proof: We sum over the negation of Condition 4 for the m_1 processors in Π_1 on which the condition fails:

$$\begin{aligned} & \sum_{\pi_k \in \Pi_1} \left(d_i - e_i < \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \right) \\ \equiv \quad & \sum_{\pi_k \in \Pi_1} (d_i - e_i) < \sum_{\tau_j \in \tau(\Pi_1)} \text{DBF}^*(\tau_j, d_i) \\ \equiv \quad & m_1 \cdot (d_i - e_i) < \sum_{\tau_j \in \tau(\Pi_1)} \text{DBF}^*(\tau_j, d_i) \end{aligned} \tag{7}$$

Since $(d_i - e_i)$ is constant within the context of the terms being summed on the right-hand side of the above inequality, Inequality 6 directly follows. ■

Lemma 3:

$$m_2 < \left(\sum_{\tau_j \in \tau(\Pi_2)} \frac{u_j}{1 - u_i} \right) \tag{8}$$

Proof: Similar to the proof of Lemma 2, we sum over the negation of Condition 5 for the m_2 processors in Π_2:

$$\begin{aligned} & \sum_{\pi_k \in \Pi_2} \left(1 - u_i < \sum_{\tau_j \in \tau(\pi_k)} u_j \right) \\ \equiv \quad & \sum_{\pi_k \in \Pi_2} (1 - u_i) < \sum_{\tau_j \in \tau(\Pi_2)} u_j \\ \equiv \quad & m_2 \cdot (1 - u_i) < \sum_{\tau_j \in \tau(\Pi_2)} u_j \end{aligned} \tag{9}$$

from which Inequality 8 follows. ■

We have thus seen that, if τ_i cannot be successfully mapped on to any processor, it must be the case that m_1 and m_2 must satisfy the upper bounds of Lemmas (2) and (3); equivalently, it must be the case that

$$m < \left(\sum_{\tau_j \in \tau(\Pi_1)} \frac{\text{DBF}^*(\tau_j, d_i)}{d_i - e_i} \right) + \left(\sum_{\tau_j \in \tau(\Pi_2)} \frac{u_j}{1 - u_i} \right)$$

Since each task τ_j, $1 \leq j < i$, is in either $\tau(\Pi_1)$ or $\tau(\Pi_2)$ (but not both), it contributes to either the first or the second term on the right hand side of the above inequality. Hence, it follows that

$$m < \sum_{j=1}^{i-1} \max \left(\frac{\text{DBF}^*(\tau_j, d_i)}{d_i - e_i}, \frac{u_j}{1 - u_i} \right) \tag{10}$$

Theorem 1 below follows, based on this inequality (and the additional observation that, since there are m processors, the first m tasks $\tau_1, \tau_2, \ldots, \tau_m$ are guaranteed to be successfully assigned):

Theorem 1: If task system τ satisfies

$$m \geq \max_{i=m+1}^{n} \left[\sum_{j=1}^{i-1} \max \left(\frac{\text{DBF}^*(\tau_j, d_i)}{d_i - e_i}, \frac{u_j}{1 - u_i} \right) \right] \tag{11}$$

then it is guaranteed to be successfully partitioned upon a multiprocessor platform comprised of m unit-capacity processors.

■

An additional observation: any task system τ with $(d_1 - e_i) = 0$ for any task $\tau_{m+1} \ldots, \tau_n$, fails the test of Theorem 1. Intuitively, what this implies is that our partitioning algorithm fails to handle systems with very rigid timing constraints on individual tasks' jobs. More generally, the larger the "slack" in each task's job's deadlines — i.e., the quantity $(d_i - e_i)$ — the more likely our algorithm is to accurately identify feasible systems.

Constrained systems ($(d_i \leq p_i)$ for all tasks τ_i)

Sporadic task systems τ, in which all tasks $\tau_i \in \tau$ satisfy the additional constraint that $d_i \leq p_i$, are sometimes referred to in the literature as **constrained** sporadic task systems.

For such constrained sporadic task systems, the condition of Theorem 1 can be further simplified to get rid of the inner "min" in Inequality 11. To see why this is so, observe that

$$\frac{\text{DBF}^*(\tau_j, d_i)}{d_i - e_i} = \frac{u_j \cdot (d_i + p_j - d_j)}{d_i - e_i} \geq \frac{u_j \cdot d_i}{d_i - e_i}$$

(since $(p_j - d_j)$ in the numerator is ≥ 0, while

$$\frac{u_j}{1 - u_i} = \frac{u_j \cdot p_i}{p_i - e_i} \; .$$

Now since $d_i \leq p_i$, it is the case that $\frac{p_i}{p_i - e_i} \leq \frac{d_i}{d_i - e_i}$, and it hence follows that

$$\frac{\text{DBF}^*(\tau_j, d_i)}{d_i - e_i} \geq \frac{u_j}{1 - u_i} \ ;$$

i.e., the quantity within the inner "min" in Inequality 11 is always given by the first term. Hence, we obtain the following corollary to Theorem 1

Corollary 1: If *constrained* task system τ satisfies

$$m \geq \max_{i=m+1}^{n} \left\lceil \frac{\sum_{j=1}^{i-1} \text{DBF}^*(\tau_j, d_i)}{d_i - e_i}, \right\rceil \quad (12)$$

then it is guaranteed to be successfully partitioned upon a multiprocessor platform comprised of m unit-capacity processors.

■

Implicit-deadline systems ($(d_i = p_i)$ for all tasks τ_i)

Recall (Section I that sporadic task systems τ, in which all tasks $\tau_i \in \tau$ satisfy the additional constraint that $d_i = p_i$, are known as *implicit-deadline* sporadic task systems. For implicit-deadline sporadic task systems, the condition of Corollary 1 can be further simplified to obtain a result that is essentially equivalent (modulo some boundary conditions) to the previous results concerning the partitioned scheduling of implicit-deadline systems that are presented in [10], [9]. This follows from the observations that $(\sum_{j=1}^{i-1} \text{DBF}^*(\tau_j, d_i))$ — the numerator of Equation 12 in Corollary 1 — can be simplified as follows:

$$\left(\sum_{j=1}^{i-1} \text{DBF}^*(\tau_j, d_i)\right) = \left(\sum_{j=1}^{i-1} u_j \cdot (d_i + p_j - d_j)\right)$$
$$= \left(\sum_{j=1}^{i-1} u_j \cdot d_i\right) = p_i \cdot \sum_{j=1}^{i-1} u_j$$

while $(d_i - e_i)$ — the denominator of Equation 12 in Corollary 1 — can be simplified as follows:

$$(d_i - e_i) = (p_i - e_i) = p_i(1 - u_i) \ .$$

Therefore, we are able to conclude that

Corollary 2: If *implicit-deadline* task system τ satisfies

$$m \geq \max_{i=m+1}^{n} \left\lceil \frac{\sum_{j=1}^{i-1} u_j}{1 - u_i}, \right\rceil \quad (13)$$

then it is guaranteed to be successfully partitioned upon a multiprocessor platform comprised of m unit-capacity processors.

■

V. An Illustrative Example

In this section, we illustrate the operation of our partitioning algorithm (and its analysis) by means of an example. The example task system we consider is presented in Table I. It consists of the ten tasks $\tau_1, \tau_2, \ldots, \tau_n$ with execution requirement, relative deadline, and minimum inter-arrival separation parameters as given, to be scheduled on a platform of three unit-capacity processors π_1, π_2, and π_3. A resulting partitioning that could be generated by our algorithm is shown on the last line of the table.

We will also apply the test of Section IV to this example task system. Observe that $d_i \leq p_i$ for each task τ_i in our example system τ; i.e., τ is a constrained-deadline task system. Hence, we may use the test of Corollary 1 to determine whether this task system would be successfully handled by our partitioning algorithm.

The quantity within the "max" on the right-hand side (rhs) of Inequality 12 is evaluated for each $i > m$ — for our example, for $i = 4, 5, \ldots, 10$. These computed values are presented in the fourth row of Table I – the details of the computation for two specific cases ($i = 4$ and $i = 7$) are elaborated on below. As can be seen from the table, the largest value here is $\leq$ the number of processors (i.e., 3); hence, the rhs of Inequality 12 is indeed $\leq m$, and this task system therefore passes the test of Corollary 1 and is deemed feasible on three processors.

We now detail the computation of the rhs of Inequality 12 for $i = 4$ and $i = 7$

- Task[4] is being considered; hence, $(\frac{\text{DBF}^*(\tau_j, d_i)}{d_i - e_i}) = \text{DBF}^*(\tau_j, d_i)/4$ must be computed for each of $j = 1, 2$, and 3, and these three largest values summed.
  - $j = 1$: $\text{DBF}^*(\tau_j, d_i)/4 = 0.75$;
  - $j = 2$: $\text{DBF}^*(\tau_j, d_i)/4 = 1$;
  - $j = 3$: $\text{DBF}^*(\tau_j, d_i)/4 = 1.03125$.

  Summing these three values: $0.75 + 1 + 1.03125 = \mathbf{2.78125}$, which is the entry in Table I
- Task[7] is being considered; hence, $\frac{\text{DBF}^*(\tau_j, d_i)}{d_i - e_i} = \text{DBF}^*(\tau_j, d_i)/9$ must be computed for each of $j = 1, 2, \ldots, 6$, and these six largest values summed.
  - $j = 1$: $\text{DBF}^*(\tau_j, d_i)/9 = 0.444444$;
  - $j = 2$: $\text{DBF}^*(\tau_j, d_i)/9 = 0.583333$;
  - $j = 3$: $\text{DBF}^*(\tau_j, d_i)/9 = 0.666667$;
  - $j = 4$: $\text{DBF}^*(\tau_j, d_i)/9 = 0.5$;
  - $j = 5$: $\text{DBF}^*(\tau_j, d_i)/9 = 0.133333$;
  - $j = 6$: $\text{DBF}^*(\tau_j, d_i)/9 = 0.266667$;

  Summing these six values: $0.444444 + 0.583333 +$

τ_i	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
e_i	2	3	3	3	1	2	3	3	2	2
d_i	2	3	4	7	8	10	12	12	14	15
p_i	10	12	8	10	20	10	12	20	20	15
$(\sum_{j=1}^{i-1} \text{DBF}^*(\tau_j, d_i))/(d_i - e_i)$	–	–	–	2.78	2.18	2.33	2.59	2.93	2.74	2.83
Proc to which assigned	π_1	π_2	π_3	π_1	π_2	π_1	π_2	π_3	π_3	π_3

TABLE I

Example task system of ten sporadic tasks, to be partitioned among three processors. (The tasks are indexed according to non-decreasing values of their relative-deadline parameters.)

$0.666667+0.5+0.1333333+0.266667 = \mathbf{2.59444}$, which is the entry in Table I.

VI. A PRAGMATIC IMPROVEMENT

We have made several approximations in deriving the results above. One of these has been the use of the approximation $\text{DBF}^*(\tau_i, t)$ of Equation 2 in Condition 4, to determine whether (the first job of) task τ_i can be accommodated on a processor π_k. We could reduce the amount of inaccuracy introduced here, by refining the approximation: rather than approximating $\text{DBF}(\tau_i, t)$ by a single step followed by a line of slope u_i, we could explicitly have included the first k steps, followed by a line of slope u_i. For the case $k = 2$, such an approximation, denoted $\text{DBF}'(\tau_i, t)$ here, is as follows:

$$\text{DBF}'(\tau_i, t) = \begin{cases} 0, & \text{if } t < d_i \\ e_i, & \text{if } d_i \le t < d_i + p_i \\ u_i \times (t + p_i - d_i), & \text{otherwise} \end{cases} \tag{14}$$

If we were to indeed use an approximation comprised of k steps, instead of the single-step approximation DBF^*, in Condition 4 in determining whether a processor can accommodate an additional task, we would need to explicitly re-check that the first k deadlines of all tasks previously assigned to the processor continue to be met. This is because it is no longer guaranteed that the new deadlines (those of τ_i) will occur *after* the deadlines of previously-assigned tasks, and hence it is possible that adding τ_i to the processor will result in some previously-added task missing one of its deadlines. However, the benefit of using better approximations is a greater likelihood of determining a system feasible; we illustrate by an example.

Example 1: Suppose that task $\tau_j = (1, 1, 10)$ has already been assigned to processor π_k when task $\tau_i = (1, 2, 20)$ is being considered. Evaluating Condition 4, we have

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

which is false; hence, we determine that τ_i fails the test of Condition 4 and cannot be assigned to processor π_k.

However, suppose that we were to instead approximate the demand bound function to two steps rather than one, by using the function $\text{DBF}'(\ , \ )$ (Equation 14 above). We would need to consider two deadlines for both the new task τ_i <u>as well as</u> the previously-assigned task τ_j. The deadlines for τ_i are at time-instants 2 and 22, and for τ_j at time-instants 1 and 11. The demand-bound computations at all four deadlines are shown below:

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

Furthermore, τ_i also passes the test of Condition 5, since $u_j + u_i = 0.1 + 0.05$ which is ≤ 1. ■

As this example illustrates, the benefit of using a finer approximation is enhanced feasibility: the feasibility test is less likely to incorrectly declare a feasible system to be infeasible. The cost of this improved performance is run-time complexity: rather than just check Condition 4 at d_i for each processor during the assignment of task τ_i, we must check a similar condition on a total of $(i \times k)$ deadlines over all m processors (observe that this remains polynomial-time, for constant k). Hence in practice, we recommend that the largest value of k that results in an acceptable run-time for the algorithm be used.

From a theoretical perspective, we were unable to obtain a significantly better bound than the one in Theorem 1 by using a finer approximation in this manner.

VII. Summary and Conclusions

Most prior theoretical research concerning the multiprocessor scheduling of sporadic task systems has imposed the additional constraint that all tasks have their deadline parameter equal to their period parameter. In this work, we have removed this constraint, and have considered the scheduling of arbitrary sporadic task systems upon preemptive multiprocessor platforms, under the partitioned paradigm of multiprocessor scheduling. We have designed an algorithm for performing the partitioning of a given collection of sporadic tasks upon a specified number of processors, and have proved the correctness of, and evaluated the effectiveness of, this partitioned algorithm. The techniques we have employed are novel and interesting, and we hope that they will be of some use in designing superior, and more efficient, algorithms for analyzing multiprocessor real-time systems.

While we have assumed in this paper that our multiprocessor platform is comprised of identical processors, we observe that our results are easily extended to apply to *uniform multiprocessor* platforms — platforms in which different processors have have different speeds or computing capacities — under the assumption that each processor has sufficient computing capacity to be able to accommodate each task in isolation. We are currently working on extending the results presented in this paper to uniform multiprocessor platforms in which this assumption may not hold.

We would also like to re-iterate that the results presented here are very pessimistic: while every system deemed to be feasible by our algorithm is guaranteed to actually be so, there are large classes of systems (e.g., those in which some tasks have their execution requirements equal to their relative deadline parameters – i.e., zero-slack tasks) which may be feasible but which our algorithm will fail to schedule. This is a consequence of the inherent intractability of the underlying problem, which is NP-hard in the strong sense.

References

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

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

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

[4] Horn, W. Some simple scheduling algorithms. *Naval Research Logistics Quarterly 21* (1974), 177–185.

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

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

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

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

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

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

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

[12] Srinivasan, A. *Efficient and Flexible Fair Scheduling of Real-time Tasks on Multiprocessors.* PhD thesis, Department of Computer Science, The University of North Carolina at Chapel Hill, 2003.