GLOBAL STATIC-PRIORITY SCHEDULING OF SPORADIC TASK SYSTEMS ON MULTIPROCESSOR PLATFORMS

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

Abstract

The multiprocessor scheduling of collections of real-time jobs is considered. Sufficient conditions are derived for determining whether a specified system meets all deadlines when scheduled by a **static-priority** algorithm. These conditions are used to obtain efficient schedulability tests for sporadic task systems scheduled using the popular Deadline-Monotonic real-time scheduling algorithm. Resource-augmentation bounds are provided that quantify the tightness of these schedulability tests.

Keywords: Multiprocessor platforms; Static-Priority Scheduling; Schedulability analysis; Resource Augmentation.

1 Introduction

Over the years, the sporadic task model [12, 6] has proven remarkably useful for the modelling of recurring processes that occur in hard-real-time systems. In this model, a *sporadic task* $\tau_k = (e_k, d_k, p_k)$ is characterized by a *worst-case execution requirement* e_k, a *(relative) deadline* d_k, and a *minimum inter-arrival separation* p_k, which is, for historical reasons, also referred to as the *period* of the task. Such a sporadic task generates a potentially infinite sequence of *jobs*, with successive job-arrivals separated by at least p_k time units. Each job has a worst-case execution requirement equal to e_k and a deadline that occurs d_k time units after its arrival time. A *sporadic task system* is comprised of several such sporadic tasks.

In this paper, we consider the scheduling of sporadic task systems upon platforms comprised of a number of identical processors, under the assumptions that jobs are preemptive, and that inter-processor migration is permitted; i.e., that a job executing upon a processor may be preempted at any instant in time, and its execution resumed later upon the same or a different processor. We will assume that a job may not be executed simultaneously on more than one processor at a time; in other words, intra-job parallelism is forbidden.

Most priority-based multiprocessor run-time scheduling algorithms operate as follows: at each instant, there is a priority assigned to each job, and the m highest-priority active jobs are selected for execution upon the m available processors. A *static-priority* algorithm for scheduling sporadic task systems associates a unique priority with each sporadic task, and all the jobs generated by a task are assigned the priority of this task. Examples of static-priority scheduling algorithms include the *rate-monotonic* algorithm [11], in which higher priorities are assigned to tasks with smaller values of the period parameter, and the *deadline-monotonic* algorithm [10], which assigns tasks priorities in inverse proportion to their relative deadline parameters: if $d_k < d_\ell$, then task τ_k has a higher priority than task τ_ℓ (ties broken arbitrarily).

Static-priority scheduling algorithms are very commonly used in real-time systems; indeed, the *Rate-Monotonic Analysis (RMA)* real-time system design and analysis methodology [9] has become a standard in the domain of hard-real-time systems. One of the benefits of scheduling systems using static-priority scheduling is that higher-priority tasks are protected from errant behavior by lower-priority tasks; hence by assigning highest priority to the most critical tasks it can be ensured that these tasks' jobs are least likely to miss their deadlines. For systems in which all tasks are equally critical, it is known that upon preemptive uniprocessors, deadline-monotonic priority assignment is optimal in the sense that if any static priority assignment will result in a system always meeting all deadlines, then so will the deadline-monotonic priority assignment.

Given the specifications of a sporadic task system and a priority assignment, (static-priority) *schedulability analysis* is the process of determining whether the system is guaranteed to always meet all deadlines if implemented upon a specified platform. Schedulability analysis of sporadic task systems when implemented upon preemptive uniprocessor platforms is well understood (see, e.g., [9]); the research reported in this document is aimed at determining schedulability-analysis algorithms for sporadic task systems which are implemented upon multiprocessor platforms.

Resource augmentation [14] provides a measure of the relative effectiveness and tightness of a given schedulability analysis algorithm. Resource augmentation works by comparing a given algorithm against the performance of a hypothetically optimal algorithm. The resource-augmentation metric of effectiveness used in this paper for a given schedulability analysis algorithm is as follows: given any task system that is always schedulable according

to the hypothetical optimal algorithm on the original platform, our goal is to obtain a constant multiplicative factor by which we must increase the speed of each processor in order for our given algorithm to achieve the same performance as the optimal algorithm. That is, we are interested in the minimum speed-up factor necessary to guarantee that our schedulability analysis algorithm verifies that a task system that is optimally schedulable on the original platform is also schedulable (according to our given non-optimal algorithm) on the more powerful, modified platform. In this paper, we obtain such resource-augmentation bounds for our schedulability analysis of deadline-monotonic scheduling.

The remainder of this paper is organized as follows. In Section 2, we formally describe our task and processor models, and discuss the motivation and applicability of these models to the modelling of actual real-time systems. In Section 3, we obtain sufficient conditions for multiprocessor schedulability analysis of real-time instances. We also obtain resource-augmentation bounds for deadline-monotonic scheduling of sporadic task systems upon a multiprocessor. In Section 4, we briefly summarize related work and place this work within a larger context of multiprocessor real-time scheduling theory.

2 Workload and processor models

Recall that each sporadic task τ_k is characterized by an execution requirement parameter e_k, a relative deadline parameter d_k, and an inter-arrival separation parameter p_k, and generates a potentially infinite sequence of jobs during run-time. Some notation: for each job J generated by some sporadic task, let $J.A$ denote its arrival time, $J.E$ its worst-case execution requirement, and $J.D$ its relative deadline: job J may need to execute for up to $J.E$ time-units over the time interval $[J.A, J.A + J.D)$ (of course, the values of $J.A$, $J.E$, and $J.D$ are determined by the parameters of the sporadic task that generated J). We also let $J.\pi$ denote the priority of the job J; the value assigned to this priority depends upon both the identity of the task that generated J and the priority assignment scheme being used.

The sporadic task model specifies minimum, rather than exact, separations between successive jobs generated by each sporadic task; therefore, any given sporadic task system may generate infinitely many different collections of jobs during different runs at run-time. We will refer to such a collection of jobs as a real-time *instance*. Specifically, a real-time instance I is comprised of a finite or infinite collection of independent jobs, that are to be executed upon a platform comprised of $m \geq 1$ identical processors.

While our primary objective in this paper is to design schedulability analysis tests for systems of sporadic tasks that are implemented upon multiprocessor platforms, some of our results (in particular, Theorem 1) are valid for arbitrary real-time instances. For a *specific* real-time instance I, schedulability analysis is the process, typically performed prior to run-time, of determining whether all jobs in I will meet their deadlines during run-time. Given the complete specifications of a finite real-time instance I, determining whether I will meet all deadlines during run-time is easily accomplished, by essentially simulating the scheduling of I.

For a sporadic task system, on the other hand, schedulability analysis is the process of determining whether all possible real-time instances that could be legally generated by this task system will meet all deadlines. Since any given sporadic task system generates infinitely many different instances, simulation of every possible real-time instance generated by a sporadic task system is not possible. To obtain schedulability analysis tests for sporadic task systems from schedulability tests for real-time instances, we must instead relate the workload characterization for real-time instances with the workload of a sporadic task system. Section 3.1 will explore the relation between the workload characterizations of real-time instances and sporadic task systems in greater detail. For the remainder of this section, we describe our workload characterization for a real-time instance I.

In what follows, we will model a real-time instance I as a finite or infinite collection of jobs: $I = \{J_1, J_2, \ldots\}$. Each job J_i is characterized by the parameters $J_i.A$, $J_i.E$, $J_i.D$ and $J_i.\pi$ as discussed above. Without loss of generality, we assume that the execution requirement parameters of jobs are normalized with respect to processor speed; i.e., $J_i.E$ denotes the amount of time for which J_i must execute. For each i, let Δ_i be defined as follows:

$$\Delta_i \stackrel{\text{def}}{=} \left(\max_{J_k.\pi \geq J_i.\pi} \{J_k.D\} \right) / J_i.D \tag{1}$$

i.e., Δ_i denotes the ratio of the largest deadline parameter of a job with priority greater than or equal to J_i's priority, to J_i's deadline parameter. This parameter will prove useful in determining schedulability conditions for real-time instances.

For any priority-level p, let us define the demand of a real-time instance over a time interval to be the sum of the execution requirements of all jobs in the instance with priority $\geq p$, that have both their arrival times and their deadlines within the interval:

$$\mathsf{demand}(p, I, t_1, t_2) \stackrel{\text{def}}{=} \sum_{(J \in I) \wedge (J.\pi \geq p) \wedge (t_1 \leq J.A) \wedge (J.A + J.D \leq t_2)} J.E \tag{2}$$

For any real-time instance I and any priority level p, let us define $\mathsf{load}(p, I)$ as follows:

$$\mathsf{load}(p, I) \stackrel{\text{def}}{=} \max_{t_1 < t_2} \frac{\mathsf{demand}(p, I, t_1, t_2)}{t_2 - t_1} \tag{3}$$

Intuitively, $\mathsf{load}(p, I)$ denotes the maximum possible cumulative computational demand, normalized by interval length, of priority p or greater that is generated by real-time instance I.

3 Sufficient schedulability conditions

We now derive sufficient conditions for determining whether a given real-time instance I is schedulable upon

a specified number of processors. In Section 3.1, we will extend these conditions to obtain sufficient schedulability conditions for the multiprocessor scheduling of sporadic task systems. In Section 3.2, we will quantify the "goodness" of these schedulability conditions via a resource-augmentation metric.

Suppose that a given real-time instance $I = \{J_1, J_2, \ldots\}$ is unschedulable upon m unit-capacity processors, and let J_i denote the first job which misses its deadline (see Figure 1). Since J_i does not receive $J_i.E$ units of execution over the interval $[J_i.A, J_i.A + J_i.D)$, it must be the case that jobs of equal or greater priority execute on all m processors for strictly more than $(J_i.D - J_i.E)$ time units during this interval. That is, such jobs of equal or greater priority execute within this interval for $> m \times (J_i.D - J_i.E)$ time units. Clearly, all of this execution belongs to jobs that arrive, and / or have their deadline in, the interval $[J_i.A, J_i.A + J_i.D)$.

Let J_s denote the job with earliest arrival time that has priority $\geq J_i.\pi$, such that $J_i.A < J_s.A + J_s.D \leq J_i.A + J_i.D$; and let J_f denote the job with latest (absolute) deadline that has priority $\geq J_i.\pi$, such that $J_i.A \leq J_f.A < J_i.A + J_i.D$ (see Figure 1). It is evident from the figure that all the execution occurring over $[J_i.A, J_i.A + J_i.D)$ is generated by jobs of priority $\geq J_i.\pi$ that arrive in, and have their deadlines within, $[J_s.A, J_f.A + J_f.D)$.

Recall, from Equation 1, that Δ_i denotes the ratio of the largest deadline parameter of a job with priority greater than or equal to J_i's priority, to J_i's deadline parameter. Hence, $J_s.D$ and $J_f.D$ are both $\leq J_i.D \times \Delta_i$. From Figure 1, it is evident that the interval $[J_s.A, J_f.A + J_f.D)$ is of length at most $(2\Delta_i + 1) \times J_i.D$. Hence for J_i to miss its deadline, it is necessary that

$$\begin{aligned}&(J_f.A + J_f.D - J_s.A)\mathsf{load}(J_i.\pi, I) - J_i.E > m(J_i.D - J_i.E)\\ &\Rightarrow (2\Delta_i + 1) \cdot J_i.D \cdot \mathsf{load}(J_i.\pi, I) - J_i.E > m(J_i.D - J_i.E)\\ &\equiv \mathsf{load}(J_i.\pi, I) > \frac{m - (m-1)\ \frac{J_i.E}{J_i.D}}{2\Delta_i + 1} \qquad (4)\end{aligned}$$

We have seen above that, in order for J_i to miss its deadline, it is necessary that Condition 4 be satisfied; Theorem 1 below follows by negating Condition 4.

Theorem 1 *Let I denote a real-time instance such that for each $J_i \in I$, the following condition is satisfied:*

$$\mathsf{load}(J_i.\pi, I) \leq \frac{1}{2\Delta_i + 1}\left(m - (m-1)\frac{J_i.E}{J_i.D}\right). \quad (5)$$

Instance I is fixed-priority schedulable upon a platform comprised of m unit-capacity processors. ■

A note. We would like to point out that while the test of Theorem 1 is sufficient for ensuring that a specified real-time instance I is schedulable, it is not necessary – an instance that fails the test of Theorem 1 may well be schedulable. Indeed, for real-time instances I for which all job parameters are a priori known, a simple exact (necessary and sufficient) schedulability test is to simulate the actual scheduling of the real-time instance. However (as discussed in Section 2 above), it is not possible to perform schedulability analysis of a sporadic task system by explicitly testing the schedulability of each of the infinitely many different real-time instances the sporadic task system could generate; hence, alternative approaches are needed to perform schedulability analysis of sporadic task systems. As we will see in Section 3.1 below, Theorem 1 provides a useful tool for designing such a schedulability test.

3.1 Sporadic Task Systems

In this section, we apply the result of Theorem 1 to obtain sufficient conditions for schedulability of real-time systems represented using the sporadic task model. Consider a sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ comprised of n tasks. Without loss of generality, ***assume that the tasks are indexed according to decreasing priorities***: task τ_1 is assigned the greatest priority, τ_n the least priority, and τ_k's priority is higher than τ_{k+1}'s for all k. We make no assumption about the relationship between a task's relative deadline and period parameter; however, if $d_k > p_k$ then it is possible for two jobs of τ_k to be active at the same time (i.e. for a given time t, two or more jobs of τ_k may have remaining execution). In this paper, we assume that different jobs of the same task may execute concurrently on different processors (i.e., while intra-job parallelism remains forbidden, intra-task parallelism is allowed). We are currently working on extending our results to systems that forbid intra-task parallelism. Below, we describe how Theorem 1 may be applied to obtain a sufficient schedulability condition for τ.

Consider all the jobs that are generated by sporadic task τ_k:

1. Observe that all such jobs J_i will have the same priority, and that Δ_i for all such jobs is given by the following:
$$\Delta_i = \frac{max_{j=1}^{k-1}(d_j)}{d_k} \qquad (6)$$

2. Notice that for all such jobs J_i generated by τ_k,
$$\frac{J_i.E}{J_i.D} = \frac{e_k}{d_k} \qquad (7)$$

3. We will determine an expression for $\mathsf{load}(J_i.\pi, I)$. For this, we make use of the concept of demand bound function [6]; for any sporadic task τ_k and any non-negative real-number t, the *demand bound function* $\mathrm{DBF}(\tau_k, t)$ denotes the maximum cumulative execution requirement that could be generated by jobs of τ_k that have both ready times and deadlines within any time interval of duration t. It is proved in [6] that
$$\mathrm{DBF}(\tau_k, t) \stackrel{\text{def}}{=} \max\left(0, \left(\left\lfloor \frac{t - d_k}{p_k} \right\rfloor + 1\right) \times e_k\right)$$
Let I denote any real-time instance that is generated during run-time by sporadic task system τ. Since the job J_i under consideration is assumed to have been

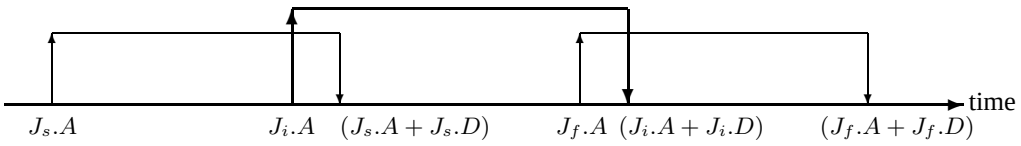

Figure 1. **Job J_i is being considered. Jobs J_s and J_f are jobs with priorities $\geq J_i.\pi$ with earliest arrival time and latest absolute deadline respectively, whose "intervals" overlap with that of J_i.**

generated by task τ_k, only tasks $\tau_1, \tau_2, \ldots, \tau_k$ will contribute to the $\mathsf{load}(J_i.\pi, I)$ term. The maximum cumulative execution requirement by jobs of these tasks over any time interval $[t_1, t_2)$ is at most the sum of the maximum execution requirements of the individual tasks:

$$\mathsf{demand}(J_i.\pi, I, t_1, t_2) \leq (\sum_{j=1}^{k} \mathrm{DBF}(\tau_j, t_2 - t_1)) \ .$$

From the definition of load (Equation 3), it follows that

$$\mathsf{load}(J_i.\pi, I) \leq \max_{t \geq 0} \left\{ (\sum_{j=1}^{k} \mathrm{DBF}(\tau_j, t))/t \right\} \quad (8)$$

Thus far, the load function has been defined with respect to real-time instances only. Let us now overload the definition of load to apply to sporadic task systems as well, as follows:

$$\mathsf{load}(k, \tau) \stackrel{\text{def}}{=} \max_{t \geq 0} \left\{ (\sum_{j=1}^{k} \mathrm{DBF}(\tau_j, t))/t \right\}$$

That is, $\mathsf{load}(k, \tau)$ denotes the maximum cumulative computational demand, normalized by interval length, of priority k or greater that can be generated by sporadic task system τ. It has been shown that load may be efficiently determined for sporadic task systems. [5, 8, 15] present algorithms that have pseudo-polynomial time complexity for task systems that have a system utilization ($\sum_{i=1}^{n} e_i/p_i$) strictly less than m. If a bounded amount of inaccuracy may be tolerated, a polynomial-time approximation scheme exists [8] which calculates load to within an arbitrary additive error term ϵ with run-time complexity of $\mathcal{O}(n^3/\epsilon)$.

Substituting from Equations 6, 7, and 8 into Theorem 1, we obtain the following theorem concerning the static-priority scheduling of sporadic task systems upon multiprocessor platforms:

Theorem 2 *Any sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ satisfying*

$$\forall k \ : 1 \leq k \leq n : \quad \mathsf{load}(k, \tau) \leq \frac{1}{2\left(\frac{\max_{j=1}^{k-1}(d_j)}{d_k}\right)+1} \left(m - (m-1)\frac{e_k}{d_k}\right) \quad (9)$$

is fixed-priority schedulable upon a platform comprised of m unit-capacity processors. ■

Recall that in deadline monotonic scheduling [10], tasks are assigned priorities in inverse proportion to their relative-deadline parameters. In that case, $\frac{\max_{j=1}^{k-1}(d_j)}{d_k}$ is always no larger than 1, and Corollary 1 below follows:

Corollary 1 *Any sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ satisfying*

$$\forall k \ : 1 \leq k \leq n : \ \mathsf{load}(k, \tau) \leq \frac{1}{3}\left(m - (m-1)\frac{e_k}{d_k}\right) \quad (10)$$

is deadline-monotonic schedulable upon a platform comprised of m unit-capacity processors. ■

3.2 Resource-augmentation bound

As mentioned in the introduction, resource augmentation quantifies the maximum multiplicative factor by which we must increase the speed of each processor to obtain the same performance as an optimal scheduling algorithm. To obtain a resource-augmentation bound for the static-priority scheduling of sporadic task systems, it is useful to derive necessary conditions for schedulability. Suppose a given condition is necessary for schedulability on an m-processor platform where each processor has speed-$1/\alpha$ (where $\alpha \geq 1$), and the same condition is *sufficient* for schedulability on a unit-capacity processor, then we may infer that the given scheduling algorithm has a resource-augmentation bound of at least α.

We can easily derive necessary conditions for sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ (with $d_k \leq d_{k+1}$ $\forall k$) to be schedulable on m unit-capacity processors. First, it is necessary that $\mathsf{load}(k, \tau) \leq m$ for all k: for each value of k, this asserts that it is possible to meet all deadlines of all jobs of priority $\geq k$. Similarly, it is necessary that $(e_k/d_k) \leq 1$ for all k (since we have assumed that execution requirements are normalized with respect to processor speed, this merely states that no job's execution requirement may exceed the time-interval between its arrival time and its deadline).

These were necessary conditions for schedulability; Corollary 2 below provides sufficient conditions:

Corollary 2 *Any sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ satisfying the following conditions*

$$\forall k \ : 1 \leq k \leq n : \ \mathsf{load}(k, \tau) \leq \frac{m^2}{4m-1} \quad (11)$$

$$\forall k \ : 1 \leq k \leq n : \ (e_k/d_k) \leq \frac{m}{4m-1} \quad (12)$$

is successfully scheduled by the deadline-monotonic scheduling algorithm upon an m-processor unit-capacity multiprocessor platform.

Proof: In order that instance I be successfully scheduled by the deadline-monotonic scheduling algorithm on m unit-capacity processors it is, by Equation 10, sufficient that for all k,

$$\begin{aligned} & \mathsf{load}(k,\tau) \leq \frac{1}{3} \times (m-(m-1)(e_k/d_k)) \\ \Leftarrow \quad & \text{(From Condition 12)} \\ & \mathsf{load}(k,\tau) \leq \frac{1}{3} \times \left(m-(m-1)\frac{m}{4m-1} \right) \\ \equiv \quad & \mathsf{load}(k,\tau) \leq \frac{m^2}{4m-1} \end{aligned}$$

which is true, (by Condition 11 above). ■

The result in Corollary 2 above can be considered to be an analog of a "utilization" test [11], or a "processor demand criteria" test [6], for uniprocessor systems. Clearly, Conditions 11 and 12 are necessary for any τ to be schedulable upon a platform comprised of m processors each of computing capacity $m/(4m-1)$; by Corollary 2 above, these conditions are also sufficient for τ to be schedulable upon m unit-capacity processors. Hence to within a constant factor of less than four, we have obtained bounds on the multiplicative speed-up needed for the conditions based on load and (e_k/d_k) to suffice for schedulability.

4 Context and Related Work

We now relate the research described in this paper to the larger body of on-going research in multiprocessor real-time scheduling theory. In Section 4.1, we briefly describe applications of the scheduling of real-time *instances*. In addition, we review some prior work on static-priority schedulability analysis for real-time instances, and relate Theorem 1 to this prior work. In Section 4.2, we review some prior related work on schedulability analysis for static-priority scheduling of *sporadic task systems*. We provide a brief comparison and discussion of the results of Corollary 1 and 2 in relation to these prior results.

4.1 The Scheduling of Real-Time Instances

To ensure a given quality-of-service, web servers may associate with each request a deadline and priority. From a different domain: real-time database systems (RTDBS) may characterize transactions by their arrival time, execution requirement, deadline, and priority [1]. In both these examples, the real-time instance characterization discussed in Section 2 can be used to model these systems. Upper bounds on the load may be obtainable through empirical or analytical methods (for example, we may obtain upper bounds for load of real-time network traffic using the well-known (σ,ρ) model [13]). Once an upper bound on load has been determined, we may apply the schedulability test of Theorem 1.

Abdelzaher et al. [2] have previously considered the static-priority multiprocessor scheduling of real-time instances. Abdelzaher et al. propose a workload characterization called *synthetic utilization*. Let $V(t)$ be the set of jobs in a real-time instance I that are active at time t (a job J_i is active at time t if $J_i.A \leq t < J_i.A + J_i.D$). The synthetic utilization of I at time t is $U_I(t) \stackrel{\text{def}}{=} \sum_{J_i \in V(t)} J_i.E/J_i.D$. An advantage of this characterization is that it may be used for efficient real-time admission-control of new jobs. However, it can be shown that any schedulability test based on only synthetic utilization has no resource-augmentation bound. Consider an instance I comprised of n jobs, all arriving at time-instant 0 and having an execution-requirement of 1, and with the i'th job's deadline at time-instant i. This instance can be successfully scheduled upon a single unit-capacity processor, yet has synthetic utilization at time zero equal to $\sum_{i=1}^{n}(1/i)$, which increases without bound with increasing n. Hence, there can be no necessary condition for schedulability based on synthetic utilization alone; thus, there is no resource-augmentation bound for pure synthetic-utilization based schedulability testing. In contrast, we may obtain an resource-augmentation bound similar to Corollary 2 for deadline-monotonic scheduling of real-time instances.

4.2 The Scheduling of Sporadic Task Systems

Recently, researchers have focussed on the multiprocessor static-priority scheduling of sporadic task systems. Baker [3, 4] and Bertogna et al. [7] have derived sufficient conditions for the multiprocessor, static-priority schedulability of sporadic task systems. The following theorem restates the schedulability test from Bertogna et al. [7]:

Theorem 3 (from [7]) *Let τ be a sporadic task system $\{\tau_1,\ldots,\tau_n\}$ ordered by decreasing priority with $d_i \leq p_i$ for each task τ_i. A task $\tau_k \in \tau$ is schedulable on m unit-capacity processors according to static-priority scheduling if*

$$\sum_{i=1}^{k-1} \min(\beta_i, 1-\frac{e_k}{d_k}) < m(1-\frac{e_k}{d_k}) \tag{13}$$

or,

$$\sum_{i=1}^{k-1} \min(\beta_i, 1-\frac{e_k}{d_k}) = m(1-\frac{e_k}{d_k}), \quad \left(\text{if } \exists i \neq k : \beta_i \leq 1-\frac{e_k}{d_k} \right) \tag{14}$$

where

$$\beta_i \stackrel{\text{def}}{=} \begin{cases} \frac{e_i}{p_i}\left(1+\frac{p_i-e_i}{d_k}\right), & \text{if } \frac{e_k}{d_k} \geq \frac{e_i}{p_i} \\ \frac{e_i}{p_i}\left(1+\frac{p_i-e_i}{d_k}\right) + \frac{e_i - p_i\frac{e_k}{d_k}}{d_k}, & \text{if } \frac{e_k}{d_k} < \frac{e_i}{p_i} \end{cases} \tag{15}$$

Previous work has only empirically evaluated the effectiveness of the approach of Theorem 3, and, to the best of our knowledge, no resource-augmentation bounds have been obtained. In the next theorem, we give a lower bound on the resource-augmentation bound associated with Theorem 3:

Theorem 4 *The schedulability tests of Theorem 3 (Conditions 13 and 14) cannot have a resource-augmentation bound smaller than* $3 - \frac{2}{m}$ *for a platform with* m*-processors (where* $m > 1$*).*

Proof: Consider the following task system τ comprised of $(m-1)\ell$ small tasks and a single large task: the small tasks $\tau_i \in \{\tau_1, \ldots \tau_{(m-1)\ell}\}$ have specification $\tau_i \stackrel{\text{def}}{=} (e_i, d_i, p_i) = (1, \ell, \ell)$; the large task $\tau_{(m-1)\ell+1} \stackrel{\text{def}}{=} (\ell, \ell, \ell)$. This task system may be scheduled on a platform with m unit-capacity processors. It is easy to see that the $(m-1)\ell$ small tasks may be scheduled upon the first $(m-1)$ processors (ℓ fit on each processor), and the large task is schedulable on the last processor.

We now determine the smallest constant $\alpha > 1$ such that $\tau_{(m-1)\ell+1}$ is schedulable according to Conditions 13 and 14 of Theorem 3. For $\tau_i \in \{\tau_1, \ldots \tau_{(m-1)\ell}\}$,

$$\beta_i = \frac{1/\alpha}{\ell}\left(1 + \frac{\ell - (1/\alpha)}{\ell}\right).$$

Taking the limit of $\ell \cdot \beta_i$ as we increase the number of small tasks, we get $\lim_{\ell\to\infty}(\ell \cdot \beta_i) = 2/\alpha$. To find the smallest α that satisfies Conditions 13 or 14 (with respect to the schedulability of $\tau_{(m-1)\ell+1}$), we must solve the following equation obtained from the conditions of Theorem 3:

$$m(1 - \frac{1}{\alpha}) \geq (m-1)\frac{2}{\alpha} \tag{16}$$

Solving for $\alpha > 1$, we find that $\alpha \geq 3 - \frac{2}{m}$ which is a lower bound on the resource-augmentation factor needed by Conditions 13 and 14. ■

The previous theorem only provides a lower bound on the resource-augmentation factor, while Corollary 2 gives an upper bound. It is interesting to note that the test of Corollary 1 applied to the example task system used in the previous proof requires a speed up of $4 - \frac{1}{m}$. Further work is needed to obtain an upper bound on the resource-augmentation factor for the test of Conditions 13 and 14 is less than that of Corollary 2.

5 Conclusions

While the static-priority scheduling of *uni*processor systems has received considerable attention, it is only recently that researchers have started to analyze multiprocessor static-priority scheduling. In this paper, we have derived very general, sufficient conditions for the schedulability of real-time instances, and have applied these conditions to obtain sufficient schedulability conditions for sporadic task systems. Prior work on multiprocessor static-priority scheduling has not considered resource-augmentation analysis. We show that our algorithm achieves a resource-augmentation bound of approximately four. Further study is required to determine whether such a bound is tight for multiprocessor static-priority scheduling.

Acknowledgements

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

References

[1] R. K. Abbott and H. Garcia-Molina. Scheduling real-time transactions: a performance evaluation. *ACM Trans. Database Syst.*, 17(3):513–560, 1992.

[2] T. Abdelzaher, B. Andersson, J. Jonsson, V. Sharma, and M. Nguyen. The aperiodic multiprocessor utilization bound for liquid tasks. In *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, San Jose, California, September 2002. IEEE Computer Society Press.

[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 fixed-priority schedulability on a multiprocessor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 32(1–2):49–71, 2006.

[5] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:301–324, 1990.

[6] 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.

[7] 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.

[8] N. Fisher, S. Baruah, and T. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Dresden, Germany, July 2006.

[9] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, Boston, 1993.

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

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

[12] 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.

[13] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Transactions on Networking*, 2(2):137–150, April 1994.

[14] 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.

[15] I. Ripoll, A. Crespo, and A. K. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 11:19–39, 1996.