3

Schedulability Analysis of Multiprocessor Sporadic Task Systems

3.1 Introduction 3-1
3.2 Definitions and Models 3-2
Sporadic Task System • A Classification of Scheduling Algorithms
3.3 Dynamic Priority Scheduling 3-4
Partitioned Scheduling • Global Scheduling
3.4 Fixed Job-Priority Scheduling 3-5
Partitioned Scheduling • Global Scheduling
3.5 Fixed Task-Priority Scheduling 3-10
Partitioned Scheduling • Global Scheduling
3.6 Relaxations of the Sporadic Model 3-13
Sleeping within a Job • Task Interdependencies • Nonpreemptability • Scheduling and Task-Switching Overhead • Changes in Task System • Aperiodic Tasks
3.7 Conclusion 3-15

Theodore P. Baker*
Florida State University
Sanjoy K. Baruah
University of North Carolina

3.1 Introduction

In this chapter, we consider the scheduling of systems of independent sporadic tasks to meet hard deadlines upon platforms comprised of several identical processors. Although this field is very new, a large number of interesting and important results have recently been obtained. We focus on presenting the results within the context of what seems to be a natural classification scheme (described in Section 3.2.2). Space considerations rule out the possibility of providing proofs of these results; instead, we provide references to primary sources where possible.

The organization of this chapter is as follows. In Section 3.2, we formally define the sporadic task model and other important concepts, and describe the classification scheme for multiprocessor scheduling algorithms that we have adopted. In each of Sections 3.3 through 3.5, we discuss results concerning one class of scheduling algorithms. The focus of Sections 3.2 through 3.5 is primarily theoretical; in Section 3.6, we briefly discuss some possible extensions to these results that may enhance their practical applicability.

*This material is based on the work supported in part by the National Science Foundation under Grant No. 0509131 and a DURIP grant from the Army Research Office.

3.2 Definitions and Models

The sporadic task model may be viewed as a set of timing constraints imposed on a more general model. A general *task* is an abstraction of a sequential thread of control, which executes an algorithm that may continue indefinitely, alternating between states where it is competing for processor time and states where it is waiting for an event. (Waiting for an event means an intentional act of the task to suspend computation, as opposed to incidental blocking such as may be imposed by contention for resources.) The computation between two wait states of a task is a *job*. The time of an event that triggers a transition from waiting to competing for execution is the *release time* of a job, and the time of the next transition to a wait state is the *completion time* of the job. The amount of processor time used by the job during this interval is its *execution time*. In hard real-time systems, a deadline is specified for a job prior to which the job's completion time must occur. The *absolute deadline* of a job is its release time plus its *relative deadline*, and the *scheduling window* of a job is the interval that begins with its release time and ends at its absolute deadline. The *response time* of a job during a particular execution of the system is defined to be the amount of elapsed time between the job's release time and its completion time. A job misses its deadline if its response time exceeds its relative deadline.

3.2.1 Sporadic Task System

Intuitively, it is impossible to guarantee that a system will not miss any deadlines if there is no upper bound on the computational demand of the system. The idea behind the sporadic task model is that if there is a lower bound on the interval between triggering events of a task and an upper bound on the amount of processor time that a task requires for each triggering event, then the computational demand of each task in any time interval is bounded.

A *sporadic task system* is a set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of sporadic tasks. Each *sporadic task* τ_i is characterized by a triple (p_i, e_i, d_i), where p_i is a lower bound on the separation between release times of the jobs of the task, also known as the *period* of the task; e_i an upper bound on the execution time for each job of the task, also known as the *worst-case execution time*; and d_i the *relative deadline*, which is the length of the scheduling window of each job. A sporadic task system is said to have *implicit deadlines* if $d_i = p_i$ for every task, *constrained deadlines* if $d_i \leq p_i$ for every task, and *arbitrary deadlines* if there is no such constraint.

A special case of a sporadic task system is a *periodic task system* in which the separation between release times is required to be exactly equal to the period.

The concepts of task and system *utilization* and *density* prove useful in the analysis of sporadic task systems on multiprocessors. These concepts are defined as follows:

Utilization: The utilization u_i of a task τ_i is the ratio e_i/p_i of its execution time to its period. The total utilization $u_{\text{sum}}(\tau)$ and the largest utilization $u_{\max}(\tau)$ of a task system τ are defined as follows:

$$u_{\text{sum}}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} u_i \qquad u_{\max}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} (u_i)$$

Constrained density: The density δ_i of a task τ_i is the ratio e_i/d_i of its execution time to its relative deadline. The total density $\delta_{\text{sum}}(\tau)$ and the largest density $\delta_{\max}(\tau)$ of a task system τ are defined as follows:

$$\delta_{\text{sum}}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \delta_i \qquad \delta_{\max}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} (\delta_i)$$

Generalized density: The generalized density λ_i of a task τ_i is the ratio $e_i / \min(d_i, p_i)$ of its execution time to the lesser of its relative deadline and its period. The total generalized density $\lambda_{\text{sum}}(\tau)$ and the largest generalized density $\lambda_{\max}(\tau)$ of a task system τ are defined as follows:

$$\lambda_{\text{sum}}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \lambda_i \qquad \lambda_{\max}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} (\lambda_i)$$

An additional concept that plays a critical role in the schedulability analysis of sporadic task systems is that of *demand bound function*. For any interval length t, the demand bound function $\mathrm{DBF}(\tau_i, t)$ of a sporadic task τ_i bounds the maximum cumulative execution requirement by jobs of τ_i that both arrive in, and have deadlines within, any interval of length t. It has been shown [12] that

$$\mathrm{DBF}(\tau_i, t) = \max\left(0, \left(\left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) e_i\right) \tag{3.1}$$

For any sporadic task system τ, a load parameter may be defined as follows:

$$\mathrm{load}(\tau) \stackrel{\mathrm{def}}{=} \max_{t>0}\left(\frac{\sum_{\tau_i \in \tau} \mathrm{DBF}(\tau_i, t)}{t}\right) \tag{3.2}$$

3.2.2 A Classification of Scheduling Algorithms

Scheduling is the allocation of processor time to jobs. An m-processor *schedule* for a set of jobs is a partial mapping of time instants and processors to jobs. It specifies the job, if any, that is scheduled on each processor at each time instant. We require that a schedule not assign more than one processor to a job at each instant, and not assign a processor to a job before the job's release time or after the job completes.

A given schedule is *feasible* for a given job set if it provides sufficient time for each job to complete within its scheduling window, that is, if the response time of each job is less than or equal to its relative deadline. A given job set is *feasible* if there exists a feasible schedule for it. In practice, feasibility does not mean much unless there is an algorithm to compute a feasible schedule for it. A job set is *schedulable* by a given scheduling algorithm if the algorithm produces a feasible schedule.

The above definitions generalize from jobs to tasks in the natural way, subject to the constraint that the jobs of each task must be scheduled serially, in order of release time and with no overlap. A sporadic task system is feasible if there is a feasible schedule for every set of jobs that is consistent with the period, deadline, and worst-case execution time constraints of the task system, and it is schedulable by a given algorithm if the algorithm finds a feasible schedule for every such set of jobs.

A *schedulability test* for a given scheduling algorithm is an algorithm that takes as input a description of a task system and provides as output an answer to whether the system is schedulable by the given scheduling algorithm. A schedulability test is *exact* if it correctly identifies all schedulable and unschedulable task systems. It is *sufficient* if it correctly identifies all unschedulable task systems, but may misidentify some schedulable systems as being unschedulable.

For any scheduling algorithm to be useful for hard-deadline real-time applications it must have at least a sufficient schedulability test that can verify that a given job set is schedulable. The quality of the scheduling algorithm and the schedulability test are inseparable, since there is no practical difference between a job set that is not schedulable and one that cannot be proven to be schedulable.

3.2.2.1 A Classification Scheme

Given a task system and a set of processors, it is possible to divide the processors into clusters and assign each task to a cluster. A scheduling algorithm is then applied locally in each cluster, to the jobs generated by the tasks that are assigned to the cluster. Each of the multiprocessor schedulability tests reported here focus on one of the following two extremes of clustering:

1. *Partitioned scheduling*: Each processor is a cluster of size one. Historically, this has been the first step in extending single-processor analysis techniques to the multiprocessor domain.
2. *Global scheduling*: There is only one cluster containing all the processors.

Observe that partitioned scheduling may be considered as a two-step process: (i) the tasks are partitioned among the processors prior to runtime; and (ii) the tasks assigned to each processor are scheduled individually during runtime, using some local (uniprocessor) scheduling algorithm.

Runtime scheduling algorithms are typically implemented as follows: at each time instant, a *priority* is assigned to each job that has been released but not yet completed, and the available processors are allocated to the highest-priority jobs. Depending upon the restrictions that are placed upon the manner in which priorities may be assigned to jobs, we distinguish in this chapter between three classes of scheduling algorithms:

1. *Fixed task-priority* (FTP) scheduling: All jobs generated by a task are assigned the same priority.
2. *Fixed job-priority* (FJP) scheduling: Different jobs of the same task may have different priorities. However, the priority of each job may not change between its release time and its completion time.
3. *Dynamic priority* (DP) scheduling: Priorities of jobs may change between their release times and their completion times.

It is evident from the definitions that FJP scheduling is a generalization of FTP scheduling, and DP scheduling is a generalization of FJP scheduling.

3.2.2.2 Predictable Scheduling Algorithms

At first glance, proving that a given task system is schedulable by a given algorithm seems very difficult. The sporadic constraint only requires that the release times of any two successive jobs of a task τ_i be separated by at least p_i time units and that the execution time of every job be in the range $[0, e_i]$. Therefore, there are infinitely many sets of jobs, finite and infinite, that may be generated by any given sporadic task set.

As observed above, partitioned scheduling algorithms essentially reduce a multiprocessor scheduling problem to a series of uniprocessor scheduling algorithms (one to each processor). Results from uniprocessor scheduling theory [29] may hence be used to identify the "worst-case" sequence of jobs that can be generated by a given sporadic task system, such that the task system is schedulable if and only if this worst-case sequence of jobs is successfully scheduled.

Ha and Liu [25,26] found a way to reduce the complexity of schedulability testing for global multiprocessor scheduling, through the concept of *predictable* scheduling algorithms. Consider any two sets $\mathcal{J}$ and $\mathcal{J}'$ of jobs that only differ in the execution times of the jobs, with the execution times of jobs in $\mathcal{J}'$ being less than or equal to the execution times of the corresponding jobs in $\mathcal{J}$. A scheduling algorithm is defined to be predictable if, in scheduling such $\mathcal{J}$ and $\mathcal{J}'$ (separately), the completion time of each job in $\mathcal{J}'$ is always no later than the completion time of the corresponding job in $\mathcal{J}$. That is, with a predictable scheduling algorithm it is sufficient, for the purpose of bounding the worst-case response time of a task or proving schedulability of a task set, to look just at the jobs of each task whose actual execution times are equal to the task's worst-case execution time.

Not every scheduling algorithm is predictable in this sense. There are well-known examples of scheduling anomalies in which shortening the execution time of one or more jobs in a task set that is schedulable by a given algorithm can make the task set unschedulable. In particular, such anomalies have been observed for nonpreemptive priority-driven scheduling algorithms on multiprocessors. Ha and Liu proved that all preemptable FTP and FJP scheduling algorithms are predictable. As a consequence, in considering such scheduling algorithms we need only consider the case where each job's execution requirement is equal to the worst-case execution requirement of its generating task.

3.3 Dynamic Priority Scheduling

In DP scheduling, no restrictions are placed upon the manner in which priorities may be assigned to jobs; in particular, the priority of a job may be changed arbitrarily often between its release and completion times.

3.3.1 Partitioned Scheduling

Upon uniprocessor platforms, it has been shown [20,29] that the earliest deadline first scheduling algorithm (uniprocessor EDF) is an optimal algorithm for scheduling sporadic task systems upon uniprocessors in the sense that any feasible task system is also uniprocessor EDF-schedulable.

As a consequence of this optimality of uniprocessor EDF, observe that any task system that is schedulable by some partitioned DP scheduling algorithm can also be scheduled by partitioning the tasks among the processors in exactly the same manner, and subsequently scheduling each partition by EDF. Since EDF is a FJP scheduling algorithm, it follows that *DP scheduling is no more powerful than FJP scheduling with respect to the partitioned scheduling of sporadic task systems.* FJP partitioned scheduling of sporadic task systems is discussed in detail in Section 3.4.1; the entire discussion there holds for DP scheduling as well.

3.3.2 Global Scheduling

3.3.2.1 Implicit-Deadline Systems

For global scheduling of implicit-deadline sporadic task systems, the following result is easily proved:

Test 1 *Any implicit-deadline sporadic task system τ satisfying*

$$u_{\text{sum}}(\tau) \leq m \quad \textit{and} \quad u_{\max}(\tau) \leq 1$$

is schedulable upon a platform comprised of m unit-capacity processors by a DP scheduling algorithm.

To see why this holds, observe that a "processor sharing" schedule in which each job of each task τ_i is assigned a fraction u_i of a processor between its release time and its deadline, would meet all deadlines—such a processor-sharing schedule may subsequently be converted into one in which each job executes on zero or one processor at each time instant by means of the technique of Coffman and Denning [18, Chapter 3].

Such an algorithm is however clearly inefficient, and unlikely to be implemented in practice. Instead, powerful and efficient algorithms based on the technique of *pfair scheduling* [11] may be used to perform optimal global scheduling of implicit-deadline sporadic task systems upon multiprocessor platforms—see Ref. 1 for a survey.

Notice that task systems τ with $u_{\text{sum}}(\tau) > m$ or $u_{\max}(\tau) > 1$ cannot possibly be feasible upon a platform comprised of m unit-capacity processors; hence, Test 1 represents an exact test for global DP-schedulability.

3.3.2.2 Constrained- and Arbitrary-Deadline Systems

To our knowledge, not much is known regarding the global DP scheduling of sporadic task systems with deadline distinct from periods. It can be shown that Test 1 is trivially extended to such systems by replacing the utilization (u) parameters by corresponding generalized density (λ) parameters:

Test 2 *Any sporadic task system τ satisfying*

$$\lambda_{\text{sum}}(\tau) \leq m \quad \textit{and} \quad \lambda_{\max}(\tau) \leq 1$$

is schedulable upon a platform comprised of m unit-capacity processors by a DP scheduling algorithm.

Unlike Test 1 above, this test is sufficient but not necessary for DP-schedulability.

3.4 Fixed Job-Priority Scheduling

3.4.1 Partitioned Scheduling

As stated in Section 3.3.1 above, the additional power of dynamic priority (DP) assignment provides no benefits with respect to partitioned scheduling; consequently, all the results in this section are valid for DP partitioned scheduling as well.

3.4.1.1 Implicit-Deadline Systems

Liu and Layland [29] proved that a periodic task system τ with implicit deadlines is schedulable on a single processor by EDF if and only if $u_{\text{sum}}(\tau) \leq 1$. They also showed that EDF scheduling is optimal for scheduling such task systems on a single processor in the sense that a task system is feasible if and only if it is schedulable by EDF. The proofs extend directly to the case of sporadic task systems with implicit deadlines.

Lopez et al. [34,35] applied this single-processor utilization bound test to partitioned multiprocessor scheduling with a first-fit-decreasing-utilization (FFDU) bin packing algorithm and obtained the following schedulability test. Although originally stated in terms of periodic tasks, the result generalizes trivially to the sporadic model.

Test 3 (FFDU EDF Utilization) *Any implicit-deadline sporadic task system τ is schedulable by some partitioned FJP algorithm (specifically, FFDU partitioning and EDF local scheduling) upon a platform of m unit-capacity processors, if*

$$u_{\text{sum}}(\tau) < \frac{m\beta_{\text{EDF}}(\tau) + 1}{\beta_{\text{EDF}}(\tau) + 1}$$

where $\beta_{\text{EDF}}(\tau) = \lfloor 1/u_{\max}(\tau) \rfloor$.

If $u_{\max}(\tau) = 1$ then $\beta_{\text{EDF}}(\tau) = 1$, and the above condition reduces to the following simpler test:

Test 4 (Simple FFDU EDF Utilization) *Any implicit-deadline sporadic task system τ is schedulable by some partitioned FJP algorithm (specifically, FFDU partitioning and EDF local scheduling) upon a platform of m unit-capacity processors, if* $u_{\text{sum}}(\tau) < (m+1)/2$.

3.4.1.2 Constrained- and Arbitrary-Deadline Systems

Baruah and Fisher [8] proposed several different algorithms for partitioning constrained- and arbitrary-deadline sporadic task systems among the processors of a multiprocessor platform. In first-fit decreasing density (FFDD) partitioning, the tasks are considered in nonincreasing order of their generalized density parameters; when considering a task, it is assigned to any processor upon which the total density assigned thus far does not exceed the computing capacity of the processor. For such a partitioning scheme, they proved the following:

Test 5 (FFDU EDF) *Any arbitrary-deadline sporadic task system τ is schedulable by a partitioned FJP algorithm (specifically, FFDD partitioning and EDF local scheduling) upon a platform comprised of m unit-capacity processors, provided*

$$\lambda_{\text{sum}}(\tau) \leq \begin{cases} m - (m-1)\lambda_{\max}(\tau) & \text{if } \lambda_{\max}(\tau) \leq \frac{1}{2} \\ \frac{m}{2} + \lambda_{\max}(\tau) & \text{if } \lambda_{\max}(\tau) \geq \frac{1}{2} \end{cases} \tag{3.3}$$

Baruah and Fisher [8] also proposed a somewhat more sophisticated partitioning scheme that runs in time polynomial (quadratic) in the number of tasks in the system. This scheme—first-fit-increasing-deadline (FFID)—considers tasks in increasing order of their relative deadline parameters. In considering a task, it is assigned to any processor upon which it would meet all deadlines using EDF as the local scheduling algorithm where the test for local EDF-schedulability is done by computing an approximation

to the DBFs of the tasks. This local EDF-schedulability test is sufficient rather than exact; with this test for local EDF-schedulability, they derived the following test:

Test 6 (FFID EDF) *Any arbitrary-deadline sporadic task system τ is schedulable by a partitioned FJP algorithm (specifically, FFID partitioning and EDF local scheduling) upon a platform comprised of m unit-capacity processors, provided*

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

For constrained-deadline systems, the test becomes somewhat simpler [9]:

Test 7 (FFID EDF; Constrained deadlines) *Any constrained-deadline sporadic task system τ is schedulable by a partitioned FJP algorithm (specifically, FFID partitioning and EDF local scheduling) upon a platform comprised of m unit-capacity processors, if*

$$\text{load}(\tau) \leq \frac{1}{2}(m - (m-1)\delta_{\max}(\tau))$$

3.4.2 Global Scheduling

3.4.2.1 Implicit-Deadline Systems

Dhall and Liu [21] showed that the uniprocessor EDF utilization bound test does not extend directly to global multiprocessor scheduling. In fact, they showed that there are implicit-deadline sporadic task systems τ with $u_{\text{sum}}(\tau) = 1 + \epsilon$ for arbitrarily small positive ϵ, such that τ is not schedulable on m processors, for any value of m. The key to this observation is that m tasks with infinitesimal utilization but short deadlines will have higher priority than a task with a longer deadline and utilization near 1. For several years, this "Dhall phenomenon" was interpreted as proof that global EDF scheduling is unsuitable for hard-real-time scheduling on multiprocessors.

Notice, however, that the phenomenon demonstrated by Dhall and Liu depends on there being at least one task with very high utilization. This observation was exploited in Refs. 24 and 44 to obtain the following sufficient condition for global EDF-schedulability of implicit-deadline sporadic task systems:

Test 8 (EDF Utilization) *Any implicit-deadline sporadic task system τ satisfying*

$$u_{\text{sum}}(\tau) \leq m - (m-1)u_{\max}(\tau) \tag{3.4}$$

is schedulable upon a platform comprised of m unit-capacity processors by the global EDF scheduling algorithm.

(Though written for periodic task systems, the proofs of the above test extend trivially to sporadic task systems with implicit deadlines.)

To handle systems with large $u_{\max}$, Srinivasan and Baruah [44] also proposed a hybrid scheduling algorithm called EDF-US[ζ], for any positive $\zeta \leq 1$. EDF-US[ζ] gives top priority to jobs of tasks with utilizations above some threshold ζ and schedules jobs of the remaining tasks according to their deadlines. In Ref. 44 it is proven that EDF-US[$m/(2m-1)$] successfully schedules any implicit-deadline sporadic task system τ with total utilization $u_{\text{sum}}(\tau) \leq m^2/(2m-1)$, regardless of the value of $u_{\max}(\tau)$. Baker [4] showed that this result implies that any sporadic task system τ with implicit deadlines can be scheduled by EDF-US[1/2] on m processors if $u_{\text{sum}}(\tau) \leq (m+1)/2$.

Goossens et al. [24] suggested another approach for dealing with cases where $u_{\max}(\tau)$ is large. The algorithm they call EDF(k) assigns top priority to jobs of the $k-1$ tasks that have the highest utilizations,

and assigns priorities according to deadline to jobs generated by all the other tasks. They showed that an implicit-deadline sporadic task system τ is schedulable on m unit-capacity processors by EDF(k) if

$$m \geq (k-1) + \left\lceil \frac{u_{\text{sum}}(\tau) - u_k(\tau)}{1 - u_k(\tau)} \right\rceil$$

where $u_k(\tau)$ denotes the utilization of the kth task when the tasks are ordered by nondecreasing utilization.

Goossens et al. [24] also proposed an algorithm called PRI-D, which computes the minimum value m such that there exists a value $k < n$ that satisfies condition (3.4.2.1) above, and then schedules the task system on m processors using EDF(k). Baker [4] suggested a variant on this idea, called EDF($k_{\min}$) here, in which the value m is fixed, k is chosen to be the minimum value that satisfies this condition, and the task system is then scheduled by EDF(k). EDF($k_{\min}$) can schedule any sporadic task system τ with implicit deadlines on m processors if $u_{\text{sum}}(\tau) \leq (m+1)/2$. This is the same utilization bound as EDF-US[1/2], but the domain of task systems that are schedulable by EDF($k_{\min}$) is a superset of those schedulable by EDF-US[1/2].

Test 9 (EDF Hybrid Utilization) *Any implicit-deadline sporadic task system τ satisfying*

$$u_{\text{sum}}(\tau) \leq (m+1)/2$$

is schedulable upon a platform comprised of m unit-capacity processors by the global scheduling algorithms EDF-US[1/2] or EDF($k_{\min}$).

Observe that this is the same bound as in partitioned FJP scheduling (Test 4).

The worst-case utilization bound of $(m+1)/2$ achieved by EDF-US[1/2] and EDF($k_{\min}$) is optimal. Andersson et al. [2] showed that the utilization guarantee for EDF or any other multiprocessor scheduling algorithm that assigns a fixed priority to each job—whether partitioned or global–cannot be higher than $(m+1)/2$ on an m-processor platform. (This result was stated for periodic task systems with implicit deadlines, but the proof applies also to the sporadic case.) Consider a set of $m+1$ identical sporadic tasks with $p_i = d_i = 2x$ and $e_i = x+1$, and consider the case when all the tasks release jobs simultaneously. It is clear that such a job set cannot be scheduled successfully on m processors. By making x sufficiently large the total utilization can be arbitrarily close to $(m+1)/2$.

3.4.2.2 Constrained- and Arbitrary-Deadline Systems

Bertogna et al. [13] showed that the results presented above for implicit-deadline systems hold for constrained-deadline systems as well, when the utilization parameters in the tests ($u_{\text{sum}}(\tau)$, $u_{\max}(\tau)$, and $u_k(\tau)$) are replaced by corresponding density parameters. Further examination of the proof reveals that it also generalizes to arbitrary deadlines. Hence,

Test 10 *Any sporadic task system τ satisfying*

$$\lambda_{\text{sum}}(\tau) \leq m - (m-1)\lambda_{\max}(\tau)$$

is schedulable upon a platform comprised of m unit-capacity processors by a fixed-priority global scheduling algorithm.

Once again, this test is sufficient but not necessary for global FJP-schedulability. Baker [3,4] determined the schedulability of some task systems more precisely by finding a separate density-based upper bound $\beta_k^{\lambda}(i)$ on the contribution of each task τ_i to the load of an interval leading up to a missed deadline of a task τ_k, and testing the schedulability of each task individually.

Test 11 (Baker EDF) *An arbitrary-deadline sporadic task system τ is schedulable on m processors using global EDF scheduling if, for every task τ_k, there exists $\lambda \in \{\lambda_k\} \cup \{u_i \mid u_i \geq \lambda_k, \tau_i \in \tau\}$, such that*

$$\sum_{\tau_i \in \tau} \min\left(\beta_k^{\lambda}(i), 1\right) \leq m(1-\lambda) + \lambda$$

where

$$\beta_k^{\lambda}(i) \stackrel{\text{def}}{=} \begin{cases} u_i\left(1 + \max\left(0, \frac{T_i - d_i}{d_k}\right)\right) & \text{if } u_i \leq \lambda \\ u_i\left(1 + \frac{p_i}{d_k}\right) - \lambda\frac{d_i}{d_k} & \text{if } u_i > \lambda \text{ and } d_i \leq p_i \\ u_i\left(1 + \frac{p_i}{d_k}\right) & \text{if } u_i > \lambda \text{ and } d_i > p_i \end{cases}$$

This test is able to verify some schedulable task systems that cannot be verified by the generalized EDF utilization test (Test 10), and the generalized EDF density bound test is able to verify some task systems that cannot be verified using the Baker test. The computational complexity of the Baker test is $\Theta(n^3)$ (where n is the number of sporadic tasks in the system) while Test 10 has an $\mathcal{O}(n^2)$ computational complexity.

A direct application of the DBF to global scheduling has not yet been obtained. However, Bertogna et al. [13] have taken a step in that direction by devising a schedulability test that uses DBF for the interval of length p_i preceding a missed deadline of some job of a task τ_k, based on exact computation of the demand of a competing task over that interval.

Test 12 (BCL EDF) *Any constrained-deadline sporadic task system is schedulable on m processors using preemptive EDF scheduling if for each task τ_k one of the following is true:*

$$\sum_{i \neq k} \min\left(\beta_k(i), 1 - \lambda_k\right) < m(1 - \lambda_k) \tag{3.5}$$

$$\sum_{i \neq k} \min\left(\beta_k(i), 1 - \lambda_k\right) = m(1 - \lambda_k) \quad \text{and} \quad \exists i \neq k : 0 < \beta_k(i) \leq 1 - \lambda_k \tag{3.6}$$

where

$$\beta_k(i) \stackrel{\text{def}}{=} \frac{DBF(\tau_i, e_k) + \min\left(e_i, \max\left(0, d_k - DBF(\tau_i, e_k)p_i/e_i\right)\right)}{d_k}$$

This test is imcomparable with both Tests 10 and 11 above in the sense that there are task systems determined to be schedulable by each test for which the other two tests fail to determine schedulability.

An alternative approach to global multiprocessor EDF schedulability analysis of sporadic task systems is based on the following result, paraphrased from Ref. 37:

Theorem 3.1 (from Ref. 37)
Any independent collection of jobs that is feasible upon a multiprocessor platform comprised of m processors each of computing capacity $m/(2m-1)$ is global EDF-schedulable upon a multiprocessor platform comprised of m unit-capacity processors.

Theorem 3.1 above may be used to transform any test demonstrating feasibility of a sporadic task system into a global FJP schedulability test. For instance, recall that Test 7 deemed any constrained-deadline sporadic task system satisfying the following condition to be schedulable by a partitioned FJP algorithm upon m unit-capacity processors:

$$\text{load}(\tau) \leq \frac{m - (m-1)\delta_{\max}(\tau)}{2}$$

Any τ satisfying the above condition is clearly feasible on m unit-capacity processors as well. This fact, in conjunction with Theorem 3.1 above, yields the following sufficient condition for multiprocessor global EDF schedulability:

Test 13 *Any constrained-deadline sporadic task system τ is global EDF-schedulable upon a platform comprised of m unit-capacity processors, provided*

$$\text{load}(\tau) \leq \frac{\frac{m^2}{2m-1} - (m-1)\delta_{\max}(\tau)}{2}$$

3.5 Fixed Task-Priority Scheduling

Partisans of EDF scheduling [16] have argued that FTP scheduling should be forgotten, since EDF scheduling has clear advantages. On single processors, the optimality of EDF ([20,29]) implies that it successfully schedules all task systems that are successfully scheduled by any FTP algorithm. Even on multiprocessors it is trivially true that FJP scheduling performs no worse than FTP scheduling since all FTP algorithms are also FJP algorithms by definition, while the converse is not true. EDF also allows each application thread to express its own deadline requirement directly, rather than indirectly through priority as in FTP-scheduled systems. In contrast, making the right priority assignments in an FTP system can be very difficult to do, and the priority assignment is fragile since a change to any thread may require adjusting the priorities of all other threads. However, there are still many loyal users of FTP scheduling, who cite its own advantages. FTP is arguably simpler than EDF to implement. It is supported by nearly every operating system, including those that support the POSIX real-time application program interface [27]. With FTP it may be simpler to design a system for overload tolerance, since the lowest-priority tasks will be impacted first. Therefore, FTP scheduling is still important and is of more immediate importance to some users than EDF.

3.5.1 Partitioned Scheduling

3.5.1.1 Implicit-Deadline Systems

Liu and Layland [29] analyzed FTP as well as EDF scheduling upon preemptive uniprocessors. They showed that ordering task priorities according to period parameters is optimal for implicit-deadline periodic task systems in the sense that if a task system is schedulable under any FTP scheme, then the task system is schedulable if tasks are ordered according to period parameters, with smaller-period tasks being assigned greater priority. This priority order is called rate monotonic (RM). The proofs extend directly to the case of sporadic tasks.

Test 14 (Uniprocessor RM Utilization) *Any implicit-deadline sporadic task system τ comprised of n tasks is schedulable upon a unit-capacity processor by RM, if $u_{\text{sum}}(\tau) \leq n(2^{1/n} - 1)$.*

Unlike the uniprocessor EDF schedulability test, this uniprocessor RM utilization test is sufficient but not necessary.

Oh and Baker [36] applied this uniprocessor utilization bound test to partitioned multiprocessor FTP scheduling using (FFDU) assignment of tasks to processors and RM local scheduling on each processor. They proved that any system of implicit-deadline sporadic tasks with total utilization $U < m(2^{1/2} - 1)$ is schedulable by this method on m processors. They also showed that for any $m \geq 2$ there is a task system with $U = (m+1)/(1+2^{1/(m+1)})$ that cannot be scheduled upon m processors using partitioned FTP

scheduling. Lopez et al. [31–33] refined and generalized this result, and along with other more complex schedulability tests, showed the following:

Test 15 (FFDU RM Utilization) *Any implicit-deadline sporadic task system τ comprised of n tasks is schedulable upon m unit-capacity processors by an FTP partitioning algorithm (specifically, FFDU partitioning and RM local scheduling), if*

$$u_{\text{sum}}(\tau) \leq (n-1)(2^{1/2}-1) + (m-n+1)(2^{1/(m-n+1)}-1)$$

3.5.1.2 Constrained- and Arbitrary-Deadline Systems

Leung [28] showed that ordering task priorities according to the relative deadline parameters (this priority assignment is called deadline monotonic [DM]) is an optimal priority assignment for the uniprocessor scheduling of constrained-deadline sporadic task systems. Hence, DM is the logical choice for use as the local scheduling algorithm in partitioned FTP schemes for the multiprocessor scheduling of constrained-deadline sporadic task systems; simulations [6] indicate that DM performs well on arbitrary-deadline sporadic task systems as well.

Fisher et al. [22] studied the partitioned FTP scheduling of sporadic constrained- and arbitrary-deadline task systems when DM scheduling is used as the local scheduler on each processor. They defined a variation on the FFID partitioning scheme described in Section 3.4.1.2, called FBB-FFD, and obtained the following schedulability tests:

Test 16 *Any arbitrary sporadic task system τ is partitioned FTP-schedulable (specifically, by FBB-FFD partitioning and DM local scheduling) upon a platform comprised of m unit-capacity processors, if*

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

For constrained-deadline systems, the test becomes somewhat simpler:

Test 17 *Any constrained-deadline sporadic task system τ is partitioned FTP-schedulable (once again, by FBB-FFD partitioning and DM local scheduling) upon a platform comprised of m unit-capacity processors, if*

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

3.5.2 Global Scheduling

3.5.2.1 Implicit-Deadline Systems

Andersson et al. [2] showed that any implicit-deadline periodic task system τ satisfying $u_{\max}(\tau) \leq m/(3m-2)$ can be scheduled successfully on m processors using RM scheduling, provided $u_{\text{sum}}(\tau) \leq m^2/(3m-1)$.

Bertogna et al. [14] proved the following tighter result:

Test 18 (RM Utilization) *Any implicit-deadline sporadic task system τ is global FTP-schedulable (by global RM scheduling) upon a platform comprised of m unit-capacity processors, if*

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

Andersson et al. [2] proposed a hybrid scheduling algorithm called RM-US$[m/(3m-2)]$, similar to EDF-US [44], that gives higher priority to tasks with utilizations above $m/(3m-2)$, and showed it is able to successfully schedule any set of independent periodic tasks with total utilization up to $m^2/(3m-1)$.

Baker [5] considered RM-US$[\zeta]$ for arbitrary values of ζ, showing that the optimum value of ζ is 1/3 and RM-US[1/3] is always successful if $u_{\text{sum}} \leq m/3$.

Bertogna et al. [14] tightened the above result and proved the following general utilization bound:

Test 19 (RM Utilization) *Any implicit-deadline sporadic task system τ is global FTP-schedulable (by global RM[1/3] scheduling) upon a platform comprised of m unit-capacity processors, if*

$$u_{\text{sum}}(\tau) \leq \frac{m+1}{3}$$

3.5.2.2 Constrained- and Arbitrary-Deadline Systems

Baker [3] derived an FTP schedulability test based on generalized density for sporadic task systems with constrained deadlines. The analysis was refined in Ref. 5 to allow arbitrary deadlines. Baker and Cirinei [7] combined an insight from Refs. 13 and 14 with the analysis of Ref. 5, and unified the treatment of FTP and FJP scheduling. If one considers only the FTP case, the Baker and Cirinei test reduces to the following:

Test 20 (BC FTP) *Any arbitrary-deadline sporadic task system τ is global FTP-schedulable upon a platform comprised of m unit-capacity processors, if for every task τ_k, there exists $\lambda \in \{\lambda_k\} \cup \{u_i \mid u_i \geq \lambda_k, \tau_i \in \tau\}$, such that one of the following is true:*

$$\sum_{i<k} \min\left(\beta_k^{\lambda}(i), 1-\lambda\right) < m(1-\lambda)$$

$$\sum_{i<k} \min\left(\beta_k^{\lambda}(i), 1-\lambda\right) = m(1-\lambda) \quad \textit{and} \quad \exists i \neq k:\ 0 < \beta_k^{\lambda}(i) \leq 1-\lambda_k$$

where

$$\beta_k^{\lambda}(i) \stackrel{\text{def}}{=} \begin{cases} u_i\left(1+\max\left(0, \frac{p_i-e_i}{d_k}\right)\right) & \textit{if } u_i \leq \lambda \\ u_i\left(1+\max\left(0, \frac{d_i+p_i-e_i-\lambda d_i/u_i}{d_k}\right)\right) & \textit{if } u_i > \lambda \end{cases}$$

The RM-US$[\zeta]$ idea can be applied to arbitrary fixed task-priority schemes and applied to task systems with arbitrary deadlines as well as task systems with implicit deadlines. With FTP-US$[\zeta]$ one gives top priority to the k tasks with utilization above ζ, for $k < m$, and schedules the remaining tasks according to some other fixed task-priority scheme. Clearly, the top priority tasks will be schedulable, since $k < m$. Any basic FTP-schedulability test can then be applied to the $n-k$ remaining tasks, using $m-k$ processors. The test may be overly conservative, but it is safe.

The idea of EDF(k) and EDF($k_{\min}$) can also be applied with FTP. That is, for FTP(k) assign top priority to jobs of the $k-1$ tasks in τ that have highest utilizations, and assign lower priorities according to the given FTP scheme to jobs generated by all the other tasks in τ. FTP($k_{\min}$) uses the least k for which FTP(k) can be verified as schedulable. The same strategy for testing schedulability using a basic FTP-schedulability test on $m-k$ processors applies.

Bertogna et al. [14] have applied to FTP scheduling the same technique they applied to FJP scheduling in Ref. 13, and obtained the following FTP analog of Test 12.

Test 21 (BCL FTP) *Any constrained-deadline sporadic task system τ is global FTP-schedulable upon a platform comprised of m unit-capacity processors, if for each task τ_k one of the following is true:*

$$\sum_{i<k} \min\left(\beta_k(i), 1-\lambda_k\right) < m(1-\lambda_k)$$

$$\sum_{i<k} \min\left(\beta_k(i), 1-\lambda_k\right) = m(1-\lambda_k) \quad \textit{and} \quad \exists i \neq k : 0 < \beta_k(i) \leq 1-\lambda_k$$

where

$$\beta_k(i) \stackrel{\text{def}}{=} \frac{N_{i,k}e_i + \min\left(e_i, \max\left(0, d_k - N_{i,k}p_i + d_i - e_i\right)\right)}{d_k}$$

and

$$N_{i,k} = \left(\left\lfloor \frac{d_k - e_i}{p_i} \right\rfloor + 1\right)$$

3.6 Relaxations of the Sporadic Model

The sporadic task model may seem very restrictive from the point of view of a programmer. The restrictions are imposed to simplify analysis, and cannot be removed entirely without making it impractical or impossible to determine whether a task system is schedulable. However, some of them can be relaxed using well-known tricks, if one is prepared to accept a degree of conservative overestimation in the analysis. The following are illustrative examples of how one can accommodate some features that at first seem to violate the sporadic model.

3.6.1 Sleeping within a Job

Within a single job, a task cannot execute for a while, put itself to sleep to wait for completion of an input or output operation, and then execute some more. That means there can be no deadlines that span more than one job.

It is possible to work around this restriction by modeling tasks with multistep jobs as if they were multiple independent jobs with appropriately related deadlines. For example, suppose a task repeats the following every 10 ms with relative deadline of 5 ms:

1. Execute computation *A*, taking up to 1 ms, then read some data from an input device.
2. Initiate an input operation, then wait for the input device to respond, where the worst-case response time of the device is 1 ms.
3. Execute computation *B*, taking up to 1 ms.
4. Initiate an output operation, and do not wait for the operation to complete.

This could be modeled by two independent tasks, each with period 10 ms. The first task would execute *A* and the initiation of the input operation, with relative deadline of 1.5 ms. The second would be released in response to the completion of the input operation (by an interrupt generated by the device) and execute *B* with deadline of 1.5 ms. If the first task can be completed within 1.5 ms, the I/O operation can be completed within 1 ms, and the second task can be completed within 1.5 ms, then the work of the original task can be completed within $1.5 + 1 + 1.5 = 4$ ms, satisfying the original relative deadline of 5 ms.

3.6.2 Task Interdependencies

The only precedence constraints in the sporadic task model are between jobs of the same task. There are no precedence constraints between jobs of different tasks, and a task cannot be blocked by a resource, such as a lock held by another task.

This restriction can be relaxed slightly when one is testing schedulability of a given task, by padding the worst-case execution time of the given task with the worst-case execution times of the longest possible combination of critical sections that can block it. However, care must be taken in a multiprocessor environment, since the worst-case impact of such resource blocking is greater than that in a single-processor environment. That is, on a single processor if a job is blocked the processor will at least be doing useful work executing another thread. On a multiprocessor, it is possible that $m - 1$ processors are idle because their tasks are blocked by a critical section of a job running on another processor. This problem may be reduced with partitioned scheduling, if tasks that contend for access to the same data are assigned to the same processor. However, in a global scheduling environment or where performance concerns dictate accessing shared data in parallel from multiple processors, contention for data access becomes unavoidable. It then becomes important to bound blocking time, by using a priority-aware locking protocol. Rajkumar et al. [38,39] have shown how to apply the priority ceiling protocol to multiprocessor systems with partitioned rate monotone (RM) scheduling, and Chen and Tripathi [17] have shown how to extend the technique to multiprocessor systems with partitioned EDF scheduling. Further study of this subject in the context of global scheduling is needed.

3.6.3 Nonpreemptability

The model assumes that a scheduler may preempt a task at any time, to give the processor to another task.

Short nonpreemptable sections may be accommodated by treating them as critical sections that can block every other task, and adding an appropriate allowance to the execution time of the task that is attempting to preempt. The problem of multiway blocking described above cannot occur. Even with global scheduling, the $m - 1$ jobs with highest priority will not be affected at all.

3.6.4 Scheduling and Task-Switching Overhead

The model does not take into account the execution time overheads involved in task scheduling, preemption, and migration of tasks between processors.

Overhead that is only incurred when a job starts or completes execution can be modeled by adding it to the execution time of the corresponding job, and overhead that is associated with timer interrupt service can be modeled by a separate task.

3.6.5 Changes in Task System

The model assumes that τ is fixed, including the number n of tasks and the values e_i, p_i, and d_i for each task.

This restriction can be relaxed by modeling the system as switching between a finite set of modes, with a different task system for each mode. If the task system for each mode is verified as schedulable, the only risk of missed deadlines is during switches between modes. Through careful control of the release times of tasks it is also possible to avoid missed deadlines during mode changes. At the extreme, one can delete the tasks of the old mode as their pending jobs complete, and wait to start up the tasks of the new mode until all of the tasks of the old mode have been deleted. As less extreme set of rules to ensure safe mode, changes in a single-processor FTP system was proposed in 1989 by Sha et al. [41], and corrected in 1992 by Burns et al. [15]. We do not know of any similar work for EDF scheduling or for FTP scheduling on multiprocessors.

3.6.6 Aperiodic Tasks

The model assumes that there is a nontrivial minimum interrelease time and that the execution time of each job of a task is consistent enough that it makes sense to schedule for the worst-case execution time.

This restriction can be relaxed by decoupling logical jobs from the jobs seen by the scheduler, and viewing the task as a server thread. That is, the thread of a sporadic server has a control structure like Figure 3.1. The server thread is scheduled according to one of the many budget-based preemptive bounded-allocation scheduling algorithms that are compatible with the underlying scheduling scheme.

```
SPORADIC_SERVER_TASK

1  for ever do
2    Sleep until queue is non-empty
3    while queue is not empty do
4        Remove next event from queue
5        Compute
```

FIGURE 3.1 Pseudocode for a sporadic server.

These scheduling schemes have in common that the scheduler maintains an execution time budget for the server thread, and it executes until either its queue is empty or the server's budget is exhausted. The server budget is replenished according to the particular scheduling algorithm. If the server is suspended for using up its budget, then, when the budget is replenished, the server resumes execution at the point it was suspended. If the replenishment and priority policies of the server limit the computational load to a level that can be modeled by a sporadic task, an aperiodic server may be treated as a sporadic task in the schedulability analysis. The sporadic server [42]*, deadline sporadic server [23], constant bandwidth server [19], and total bandwidth server [43] are examples of such scheduling policies. For an overview of simple aperiodic server scheduling algorithms for the FTP and EDF environments, see Ref. 40, and for more detail see Ref. 30. For specifics on how this approach can be applied to a constant bandwidth server (CBS) on a multiprocessor with EDF scheduling, see the work of Baruah et al. [10].

It appears that multiprocessor applications of aperiodic server scheduling, combined with bandwidth recovery techniques to compensate for overallocation of processing resources to guarantee hard deadline tasks, have a strong potential to exploit the advantages of global scheduling on multiprocessors.

3.7 Conclusion

We have shown how demand analysis techniques developed originally for the analysis of schedulability with priority-based preemptive scheduling on single-processor systems have been extended to apply to multiprocessor systems. There are several practical algorithmic tests for verifying the schedulability of such systems under both EDF and fixed task-priority scheduling in both partitioned and global modes. While none of the multiprocessor scheduling algorithms we have considered here is optimal, and none of the corresponding schedulability tests is tight, experiments with pseudorandomly generated task sets have shown that some of them are successful on most task sets, up to nearly full system utilization.

It is likely that further improvements can still be made in the accuracy of schedulability tests for global scheduling on multiprocessors, with fixed job-priority schemes like EDF and FTP, and that more dramatic improvements may be made as research on dynamic job-priority schemes progresses. However, the present state of knowledge is already sufficient to make effective use of multiple processors for systems with hard deadlines.

References

1. J. Anderson, P. Holman, and A. Srinivasan. Fair multiprocessor scheduling. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press LLC, Boca Raton, FL, 2003.

*The original description of the sporadic server in Ref. 42 contained a defect, that led to several variations. See Ref. 30 for details.

2. B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proc. 22nd IEEE Real-Time Systems Symposium*, pages 193–202, London, December 2001.
3. T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proc. 24th IEEE Real-Time Systems Symposium*, pages 120–129, 2003.
4. T. P. Baker. An analysis of EDF scheduling on a multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 15(8):760–768, 2005.
5. T. P. Baker. An analysis of fixed-priority scheduling on a multiprocessor. *Real-Time Systems*, 32(1–2):49–71, 2005.
6. 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, Tallahassee, FL, July 2005.
7. T. P. Baker and M. Cirinei. A unified analysis of global EDF and fixed-task-priority schedulability of sporadic task systems on multiprocessors. Technical Report TR-060501, Department of Computer Science, Florida State University, Tallahassee, FL, May 2006.
8. S. Baruah and N. Fisher. Partitioned multiprocessor scheduling of sporadic task systems. In *Proc. 26th IEEE Real-Time Systems Symposium*, Miami, FL, December 2005. IEEE Computer Society Press.
9. S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers*, 55(7):918–923, 2006.
10. S. Baruah, J. Goossens, and G. Lipari. Implementing constant-bandwidth servers upon multiprocessor platforms. In *Proc. 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 154–163, San Jose, CA, September 2002.
11. S. K. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.
12. S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proc. 11th IEEE Real-Time Systems Symposium*, pages 182–190, 1990.
13. M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proc. 17th Euromicro Conference on Real-Time Systems*, pages 209–218, Palma de Mallorca, Spain, July 2005.
14. M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *Proc. 9th International Conference on Principles of Distributed Systems*, Pisa, Italy, December 2005.
15. A. Burns, K. Tindell, and A. J. Wellings. Mode changes in priority preemptive scheduled systems. In *Proc. 13th IEEE Real-Time Systems Symposium*, pages 100–109, Phoenix, AZ, 1992.
16. G. Buttazzo. Rate-monotonic vs. EDF: Judgement day. *Real-Time Systems*, 29(1):5–26, 2005.
17. C. M. Chen and S. K. Tripathi. Multiprocessor priority ceiling protocols. Technical Report CS-TR-3252, Department of Computer Science, University of Maryland, April 1994.
18. J. E. Coffman and P. J. Denning. *Operating Systems Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1973.
19. Z. J. Deng, W. S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *Proc. 9th Euromicro Workshop on Real-Time Systems*, pages 191–199, June 1997.
20. M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *Proc. IFIP Congress*, pages 807–813, 1974.
21. S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1): 127–140, 1978.
22. N. Fisher, S. Baruah, and T. Baker. The partitioned scheduling of sporadic tasks according to static priorities. In *Proc. EuroMicro Conference on Real-Time Systems*, pages 118–127, Dresden, Germany, July 2006. IEEE Computer Society Press.
23. T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9, pages 31–67, 1995.
24. J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2–3):187–205, 2003.

25. R. Ha. *Validating timing constraints in multiprocessor and distributed systems*. PhD thesis, University of Illinois, Department of Computer Science, Urbana-Champaign, IL, 1995.
26. R. Ha and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proc. 14th IEEE International Conference on Distributed Computing Systems*, pages 162–171, Poznan, Poland, June 1994. IEEE Computer Society.
27. IEEE Portable Applications Standards Committee. *ISO/IEC 9945-2:2003(E), Information Technology–Portable Operating System Interface (POSIX) – Part 2: System Interfaces*. IEEE Standards Association, 2003.
28. J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation (Netherlands)*, 2(4):237–250, 1982.
29. C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
30. J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, Upper Saddle River, NJ, 2000.
31. J. M. Lopez, J. L. Diaz, and D. F. Garcia. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In *Proc. 13th Euromicro Conference on Real-Time Systems*, pages 67–75, Delft, Netherlands, June 2001.
32. J. M. Lopez, J. L. Diaz, and D. F. Garcia. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(7): 642–653, 2004.
33. 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.
34. J. M. Lopez, J. L. Diaz, M. Garcia, and D. F. Garcia. Utilization bounds for multiprocessor RM scheduling. *Real-Time Systems*, 24(1):5–28, 2003.
35. J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proc. 12th Euromicro Conference on Real-Time Systems*, pages 25–33, 2000.
36. D. I. Oh and T. P. Baker. Utilization bounds for N-processor rate monotone scheduling with stable processor assignment. *Real-Time Systems*, 15(2):183–193, 1998.
37. C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, TX, 1997. ACM.
38. R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proc. 9th IEEE Real-Time Systems Symposium*, pages 259–269, 1988.
39. R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for shared memory multiprocessors. In *Proc. 10th International Conference on Distributed Computing*, pages 116–125, 1990.
40. L. Sha, T. Abdelzaher, K. E. Årzén, A. Cervin, T. P. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2–3):101–155, 2004.
41. L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):244–264, 1989.
42. B. Sprunt, L. Sha, and L. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.
43. M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2):179–210, 1996.
44. A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84:93–98, 2002.