A Survey of Hard Real-Time Scheduling for Multiprocessor Systems

ROBERT I. DAVIS and ALAN BURNS, University of York

35

This survey covers hard real-time scheduling algorithms and schedulability analysis techniques for homogeneous multiprocessor systems. It reviews the key results in this field from its origins in the late 1960s to the latest research published in late 2009. The survey outlines fundamental results about multiprocessor real-time scheduling that hold independent of the scheduling algorithms employed. It provides a taxonomy of the different scheduling methods, and considers the various performance metrics that can be used for comparison purposes. A detailed review is provided covering partitioned, global, and hybrid scheduling algorithms, approaches to resource sharing, and the latest results from empirical investigations. The survey identifies open issues, key research challenges, and likely productive research directions.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: *Real-time and embedded systems*

General Terms: Performance, Design, Algorithms

Additional Key Words and Phrases: Hard real-time scheduling, global scheduling, partitioned scheduling, multiprocessor, multicore

ACM Reference Format:
Davis, R. I. and Burns, A. 2011. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43, 4, Article 35 (October 2011), 44 pages.
DOI = 10.1145/1978802.1978814 http://doi.acm.org/10.1145/1978802.1978814

1. BACKGROUND

Today, real-time embedded systems are found in many diverse application areas, including automotive electronics, avionics, telecommunications, space systems, medical imaging, and consumer electronics. In all of these areas, there is rapid technological progress.

Companies building embedded real-time systems are driven by a profit motive. To succeed, they aim to meet the needs and desires of their customers by providing systems that are more capable, more flexible, and more effective than those of their competitors, and by bringing these systems to market earlier. This desire for technological progress has resulted in a rapid increase in both software complexity and the processing demands placed on the underlying hardware.

To address demands for increasing processor performance, silicon vendors no longer concentrate wholly on the miniaturization needed to increase processor clock speeds, as this approach has led to problems with both high power consumption and excessive

This work has been funded in part by the EU FP7 projects Jeopard (project number 216682) and eMuCo (project number 216378), the EU FP7 Artist Design Network of Excellence, and the EPSRC project Tempo (EP/G055548/1).
Authors' addresses: R. I. Davis and A. Burns, Real-Time Systems Research Group, Department of Computer Science, University of York, York, YO10 5DG, U.K.; email: {rob.davis, alan.burns}@cs.york.ac.uk.

DOI 10.1145/1978802.1978814 http://doi.acm.org/10.1145/1978802.1978814

heat dissipation. Instead, there is now an increasing trend toward using multiprocessor platforms for high-end real-time applications.

A key date in the move toward multiprocessor systems was May 7, 2004 [Bertogna 2007], when Intel canceled the successor to the Pentium P4 processor called *Tejas*, due to extremely high power consumption [Lammers 2004]. Dynamic power consumption, the power lost charging and discharging capacitive load, is a dominant factor for chip designs using technology above the 100-nm level; however, for sub-100-nm technology, transistor leakage current becomes important. This is because, as the dimensions of the gates and oxide layers are reduced, so is their electrical resistance. The result is that leakage current and hence power dissipation rapidly increases with further miniaturization. This problem can be partially ameliorated by running at a lower voltage, which reduces power consumption due to both dynamic and leakage sources; however, reducing voltage also limits the maximum operating frequency, restricting performance. A solution to this problem is to limit miniaturization and operating frequencies, and, instead, to use multiple processors on a single chip. On July 27, 2006, 2 years after the cancellation of Tejas, Intel released the Core Duo processor. In future, it is expected that the high-end processing performance will be provided by using a large number of processor cores on a single chip. For example, the Intel Teraflop Research Chip (Polaris), announced on Feb 11, 2007, has 80 processor cores providing 1 Teraflop performance at 3 GHz.

Multicore processors from other vendors include, from AMD: Opteron, Phenom, Turion 64, Radeon, and Firestream; from Analog Devices: Blackfin; from Azul Systems: Vega 1, Vega 2, Vega 3; from ARM: MPCore; from Cavium Networks: Octeon; from Freescale Semiconductor: QorlQ; from IBM: POWER4, POWER5, POWER6, PowerPC970, Xenon (X-Box 360); from Intel: Core Duo, Core 2 Duo, Core 2 Quad, Core i3, i5, i7, i7 Extreme Edition family, Itanium 2, Pentium D, Pentium Dual-Core, Polaris, Xeon; from Nvidia: GeForce 9, GeForce 200, Tesla; from NXP Nexperia; from Sun Microsystems: MAJC 5200, UltraSPARC IV, UltraSPARC T1, UltraSPARC T2; from Texas Instruments: TMS320C80 MVP; from Tilera: TILE64; from XMOS: XS-G4.

By 2009, there was clear evidence of a strong trend toward using multicore processors in embedded systems that require hard real-time performance. Chip Downing, Director of Business Development, A&D, at WindRiver, noted in the October 2009 edition of *Avionics* magazine, "We see a strong trend towards multicore in aerospace and defense" [Downing 2009]. The same article explained that "the advantage for avionics is that multicore chips can significantly reduce the Size, Weight and Power (SWaP) requirements" [Downing 2009]. A similar trend is also evident in automotive electronics, [Leteinturier 2007], and in space/satellite systems, where the European Space Agency (ESA) is supporting development of suitable multicore processors (LEON3). Mobile phones and related devices already exploit multiprocessor platforms.

1.1. Multiprocessor Real-Time Systems and Scheduling

Systems are referred to as *real-time* when their correct behavior depends not only on the operations they perform being logically correct, but also on the time at which they are performed. For example in avionics, flight control software must execute within a fixed time interval in order to accurately control the aircraft. In automotive electronics there are tight time constraints on engine management and transmission control systems that derive from the mechanical systems that they control.

Guaranteeing real-time performance while making the most effective use of the available processing capacity requires the use of efficient scheduling policies or algorithms supported by accurate schedulability analysis techniques. These analysis techniques need to be capable of analyzing the worst-case behavior of the application under a given scheduling policy, thus providing proof, subject to a set of assumptions about

application behavior, that timing constraints will always be met during operation of the system.

Research into uniprocessor real-time scheduling can trace its origins back to the late 1960s and early 1970s with significant research effort and advances made in the 1980s and 1990s. The interested reader is referred to Audsley et al. [1996] and Sha et al. [2004], which provide historical accounts of the most important advances in the field of uniprocessor scheduling during those decades. Today, although there is still significant scope for further research, uniprocessor real-time scheduling theory can be viewed as reasonably mature, with a large number of key results documented in textbooks such as those by Burns and Wellings [2009], Buttazzo [2005], and Liu [2000], and successfully transferred into industrial practice.

Multiprocessor real-time scheduling theory also has it origins in the late 1960s and early 1970s. Liu [1969, p. 28] noted that multiprocessor real-time scheduling is intrinsically a much more difficult problem than uniprocessor scheduling:

> "Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors."

The seminal paper of Dhall and Liu [1978] heavily influenced the course of research in this area for two decades. During the 1980s and 1990s, conventional wisdom was that global approaches to multiprocessor scheduling, where tasks may migrate from one processor to another, suffered from the so-called "Dhall effect," and were therefore inferior to partitioned approaches, with a fixed allocation of tasks to processors. Research efforts therefore focused almost exclusively on partitioned approaches. It was not until Phillips et al. [1997] showed that the Dhall effect was more of a problem with high-utilization tasks than it was with global scheduling algorithms that there was renewed interest in global scheduling algorithms.

In the late 1990s silicon vendors such as IBM and AMD began research into the development of multicore processors, with IBM releasing the first nonembedded dual-core processor, the POWER4, in 2001. This trend away from increasing processing capacity via ever higher clock speeds toward increasing performance via multiple processor cores became evident to the real-time systems research community. This resulted in significant research effort being focused on the problem of real-time multiprocessor scheduling. While markedly more articles have been published in this area since 2000 than before, and significant progress has been made, there are still many open questions and research challenges that remain.

This article presents a survey of multiprocessor real-time scheduling algorithms and schedulability analysis techniques, from the origins of the field in the late 1960s up to the latest research published at the end of 2009. The aim of the survey is to provide a classification of existing research, both providing a perspective on the area and identifying significant open issues and future research directions.

1.2. Organization

The remainder of the article is organized as follows: Section 2 provides a classification of multiprocessor systems, and algorithms. It describes the basic system and task models, and defines the terminology and notation used. Section 3 describes metrics that can be used to compare the performance of different multiprocessor real-time scheduling algorithms and their analyses. Section 4 describes a set of fundamental results that are independent of specific scheduling algorithms. This is followed by an overview of partitioned and global approaches to multiprocessor real-time scheduling, in Sections 5 and 6, respectively. Section 7 outlines hybrid approaches that attempt to combine the

best attributes of both partitioned and global approaches. Section 8 describes research into protocols and analyses for accessing mutually exclusive shared resources. Section 9 reports on the latest empirical research. Finally, Section 10 identifies key open issues in the field, and Section 11 provides some concluding remarks.

2. SYSTEM MODELS, TERMINOLOGY, AND NOTATION

This section provides a primer on the terminology and notation used in multiprocessor scheduling research. It is aimed both at helping new researchers entering the field and providing a consistent nomenclature that has yet to fully emerge from the research community.

2.1. Classification of Multiprocessor Systems

From the perspective of scheduling, multiprocessor systems can be classified into three categories.

(1) *Heterogeneous*. The processors are different; hence the rate of execution of a task depends on both the processor and the task. Indeed, not all tasks may be able to execute on all processors.
(2) *Homogeneous*. The processors are identical; hence the rate of execution of all tasks is the same on all processors.
(3) *Uniform*. The rate of execution of a task depends only on the speed of the processor. Thus a processor of speed 2 will execute all tasks at exactly twice the rate of a processor of speed 1.

In this survey, we are concerned with *homogeneous* multiprocessor systems, comprising m processors.

2.2. Periodic and Sporadic Task Models

The aim of multiprocessor real-time scheduling is to execute the set of tasks that make up an application, on the multiprocessor system, such that their time constraints are always met. An application (or *taskset* τ) is assumed to comprise a static set of n tasks ($\tau_1 \cdots \tau_n$). When fixed-priority scheduling is used, the task number is also used to indicate a unique priority i, from 1 to n (where n is the lowest priority).

The overwhelming majority of the research into multiprocessor real-time scheduling focuses on two simple task models: the *periodic task model* and the *sporadic task model*. In both models, tasks give rise to a potentially infinite sequence of invocations (or *jobs*). In the periodic task model, the jobs of a task arrive strictly periodically, separated by a fixed time interval. In the sporadic task model, each job of a task may arrive at any time once a minimum interarrival time has elapsed since the arrival of the previous job of the same task.

Periodic tasksets may be classified as *synchronous* if there is some point in time at which all of the tasks arrive simultaneously, or *asynchronous*, where task arrival times are separated by fixed offsets and there is no simultaneous arrival time. In the sporadic task model, the arrival times of the jobs of different tasks are assumed to be independent.

Intratask parallelism is not permitted by either model; hence, at any given time, each job may execute on at most one processor. Also, it is assumed, unless otherwise stated, that only a single job of a task is ready to execute at any given time. Further, it is assumed that once a job starts to execute it will not suspend itself.

Each task τ_i is characterized by: its relative *deadline* D_i, *worst-case execution time* C_i, and minimum interarrival time or *period* T_i. The *utilization* u_i, of task τ_i is given by C_i/T_i. The utilization u_{sum} of a taskset is the sum of the utilizations of all of its tasks. A task's *worst-case response time* R_i is defined as the longest time from a job of that

task arriving to it completing execution. The *hyperperiod* $H(\tau)$ of a taskset is defined as the least common multiple of the task periods.

There are three levels of constraint on task deadlines that are studied in the literature.

(1) *Implicit deadlines*. All task deadlines are equal to their periods ($D_i = T_i$).
(2) *Constrained deadlines*. All task deadlines are less than or equal to their periods ($D_i \leq T_i$).
(3) *Arbitrary deadlines*. Task deadlines may be less than, equal to, or greater than their periods.

Most of the published research assumes that tasks are independent and so cannot be *blocked* from executing by another task other than due to contention for the processors. Section 8 outlines research into policies that permit access to mutually exclusive resources lifting the restriction of independence. They consider the *blocking time* during which tasks can be prevented from executing due to other tasks accessing mutually exclusive shared resources.

As a result of preemption and subsequent resumption, a job may, in the case of global scheduling, migrate from one processor to another. The cost of preemption, migration, and the runtime operation of the scheduler is generally assumed to be either negligible, or subsumed into the worst-case execution time of each task. Empirical research considering the effects of such overheads is outlined in Section 9.

2.3. Taxonomy of Multiprocessor Scheduling Algorithms

Multiprocessor scheduling can be viewed as attempting to solve two problems.

(1) The *allocation problem*, or on which processor a task should execute.
(2) The *priority problem*, or when, and in what order with respect to jobs of other tasks, each job should execute.

Scheduling algorithms for multiprocessor systems can be classified according to when changes to priority and allocation can be made (referred to as *migration-based* and *priority-based* classifications [Carpenter et al. 2004]).

Allocation

(1) *No migration*. Each task is allocated to a processor and no migration is permitted.
(2) *Task-level migration*. The jobs of a task may execute on different processors; however, each job can only execute on a single processor.
(3) *Job-level migration*. A single job can migrate to and execute on different processors; however, parallel execution of a job is not permitted.

Priority

(1) *Fixed task priority*. Each task has a single fixed priority applied to all of its jobs.
(2) *Fixed job priority*. The jobs of a task may have different priorities, but each job has a single static priority. An example of this is earliest deadline first (EDF) scheduling.
(3) *Dynamic priority*. A single job may have different priorities at different times, for example least laxity first (LLF) scheduling.

Scheduling algorithms where no migration is permitted are referred to as *partitioned*, those where migration is permitted are referred to as *global*. As the majority of research into global scheduling algorithms has focused on models where arbitrary migration (job-level migration) is permitted, in the remainder of this article we will use the term *global* to mean job-level migration and provide clarification indicating when only task-level migration is permitted.

A scheduling algorithm is said to be *work-conserving* if it does not permit there to be any time at which a processor is idle and there is a task ready to execute. Partitioned scheduling algorithms are not work-conserving, as a processor may become idle, but cannot be used by ready tasks allocated to a different processor.

Scheduling algorithms can be further classified as follows.

(1) *Preemptive*. Tasks can be preempted by a higher priority task at any time.
(2) *Nonpreemptive*. Once a task starts executing, it will not be preempted and will therefore execute until completion.
(3) *Cooperative*. Tasks may only be preempted at defined scheduling points within their execution. Effectively, execution of a task consists of a series of nonpreemptable sections.

In this survey, we focus only on preemptive scheduling algorithms.

2.4. Schedulability, Feasibility, and Optimality

A taskset is said to be *feasible* with respect to a given system if there exists some scheduling algorithm that can schedule all possible sequences of jobs that may be generated by the taskset on that system without missing any deadlines.

A scheduling algorithm is said to be *optimal* with respect to a system and a task model if it can schedule all of the tasksets that comply with the task model and are feasible on the system.

A scheduling algorithm is said to be *clairvoyant* if it makes use of information about future events, such as the precise arrival times of sporadic tasks, or actual execution times, which are not generally known until they happen.

A task is referred to as *schedulable* according to a given scheduling algorithm if its worst-case response time under that scheduling algorithm is less than or equal to its deadline. Similarly, a taskset is referred to as schedulable according to a given scheduling algorithm if all of its tasks are schedulable.

A schedulability test is termed *sufficient*, with respect to a scheduling algorithm and a system if all of the tasksets that are deemed schedulable according to the test are in fact schedulable. Similarly, a schedulability test is termed *necessary* if all of the tasksets that are deemed unschedulable according to the test are in fact unschedulable. A schedulability test that is both sufficient and necessary is referred to as *exact*.

2.5. Processor Demand Function

The concepts of processor *demand bound function* $h(t)$ and processor *load* [Baruah et al. 1990a, 1990b] are used extensively in the analysis of multiprocessor scheduling. The processor demand bound function $h(t)$ corresponds to the maximum amount of task execution that can be released in a time interval $[0, t)$ and also has to complete in that interval.

$$h(t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i. \tag{1}$$

The processor *load* is the maximum value of the processor demand bound divided by the length of the time interval.

$$load(\tau) = \max_{\forall t}\left(\frac{h(t)}{t}\right). \tag{2}$$

As a taskset cannot possibly be schedulable according to any algorithm if the total execution that is released in an interval and must also complete in that interval exceeds the available processing capacity, the processor load provides a simple necessary

Table I. Notation

Symbol	Description
τ_i	Task i at priority level i
B_i	Blocking time at priority level i
C_i	Worst-case execution time of task τ_i
D_i	Relative deadline of task τ_i
δ_i	Density of task τ_i,$\delta_i = C_i / \min(D_i, T_i)$
$\delta_{\max}$	Maximum density of any task in the taskset
δ_{sum}	Taskset density (sum of task densities)
f_A	Speedup factor (resource augmentation factor) for scheduling algorithm A
$h(t)$	Processor demand in the interval [0,t)
$H(\tau)$	Hyperperiod of the taskset
$load(\tau)$	Processor load of taskset τ
$load(\tau, k)$	Processor load of taskset τ, due to tasks of priority higher than or equal to k
m	Number of processors
$M_A(\tau)$	Minimum number of processors needed to schedule taskset τ using scheduling algorithm A
n	Number of tasks
N	Number of jobs (typically in the hyperperiod of the taskset)
R_i	Worst-case response time of task τ_i
$\Re_A$	Approximation ratio for scheduling algorithm A
t	Time
T_i	Minimum interarrival time of task τ_i
u_i	Utilization of task τ_i
$u_{\max}$	Maximum utilization of any task in the taskset
u_{sum}	Taskset utilization
U_A	Utilization upper bound for scheduling algorithm A

condition for taskset feasibility [Baruah and Fisher 2005]:

$$load(\tau) \leq m, \tag{3}$$

where m is the number of processors.

2.6. Notation

For ease of reference, Table I provides a summary of the notation used in the rest of the article. Note that some of this notation refers to concepts introduced in subsequent sections. This notation has been chosen to reflect common usage. Standardizing on a common notation such as this would ease communication of results among the research community.

3. PERFORMANCE METRICS

In this section, we describe four performance metrics that have been used to compare the effectiveness of different multiprocessor scheduling algorithms and schedulability analyses. These are (i) utilization bounds, (ii) approximation ratios, (iii) resource augmentation or speedup factors, and (iv) empirical measures, such as the percentage of tasksets that are found to be schedulable.

3.1. Utilization Bounds

For implicit-deadline tasksets, worst-case *utilization bounds* are a useful performance metric. The worst-case utilization bound U_A for a scheduling algorithm A is defined as the minimum utilization of any implicit-deadline taskset that is only just schedulable according to algorithm A. Hence there exist implicit-deadline tasksets with total utilization infinitesimally greater than U_A that are unschedulable according to algorithm A. Conversely, there are no implicit-deadline tasksets with total utilisation $u_{sum} \leq U_A$

that are unschedulable according to algorithm A. Hence U_A can be used as a simple sufficient, but not necessary, schedulability test.

3.2. Approximation Ratio

The *approximation ratio* is a way of comparing the performance of a scheduling algorithm A with that of an optimal algorithm. For example, consider the problem of determining the minimum number of processors required to schedule a given taskset τ. Let the number of processors required according to an optimal algorithm be $M_O(\tau)$ and the number required according to algorithm A be $M_A(\tau)$; then the approximation ratio $\Re_A$ of algorithm A is given by

$$\Re_A = \lim_{M_O(\tau)\to\infty} \left(\max_{\forall \tau} \left(\frac{M_A(\tau)}{M_O(\tau)} \right) \right). \tag{4}$$

Note that $\Re_A \geq 1$, with smaller values of the approximation ratio indicative of a more effective scheduling algorithm, and $\Re_A = 1$ implying an optimal algorithm. Scheduling algorithms are referred to as *approximate* if they have a finite approximation ratio.

3.3. Resource Augmentation

The *resource augmentation* factor [Kalyanasundaram, and Pruhs 1995] is an alternative method of comparing the performance of a scheduling algorithm A with that of an optimal algorithm. Rather than considering the increased number of processors that would be required to obtain schedulability under algorithm A, the resource augmentation factor instead considers the increase in processing speed that would be required, assuming a linear decrease in task execution times with processing speed.

The resource augmentation or *speedup factor* f for a scheduling algorithm A is defined as the minimum factor by which the speed of all m processors would need to be increased such that all tasksets that are feasible (i.e., schedulable according to an optimal scheduling algorithm) on m processors of speed 1 become schedulable under algorithm A.

Let τ be a taskset that is feasible on a system of m processors of unit processing speed. Now assume that, using scheduling algorithm A, taskset τ is just schedulable on a system of m processors, each of speed $f(\tau)$. The resource augmentation or *speedup factor* f_A for algorithm A is given by

$$f_A = \max_{\forall m, \forall \tau} (f(\tau)) \tag{5}$$

Note that $f_A \geq 1$, with smaller values indicative of a more effective algorithm, and $f_A = 1$ implying an optimal algorithm.

3.4. Empirical Measures

A comparative measure of the effectiveness of different scheduling algorithms and their analyses can be obtained by evaluating the number of randomly generated tasksets that each deems schedulable. Ideally, the number of tasksets deemed schedulable by a schedulability test would be compared to the number of feasible tasksets generated; however, as exact feasibility tests are not known for the case of sporadic tasksets and are potentially intractable for periodic tasksets, researchers have typically used this empirical measure to compare the relative performance of two or more sufficient schedulability tests/scheduling algorithms. In these empirical comparisons, it is important to use a taskset generation algorithm that is unbiased [Bini and Buttazzo 2005], and ideally one that allows tasksets to be generated that comply with a specified parameter setting. That way the dependency of schedulability test effectiveness on each

taskset parameter can be examined by varying that parameter, while holding all other parameters constant, thus avoiding any confounding effects.

Other useful empirical techniques used by researchers include simulation of the schedule produced by different algorithms to determine the number of preemptions and migrations. While simulation cannot in general prove schedulability, it can prove that a taskset is unschedulable if the simulation reveals a deadline miss. Hence simulation can also be used as a sufficient test of unschedulability.

4. FUNDAMENTAL RESULTS

In this section, we describe a set of fundamental results about multiprocessor real-time scheduling that are independent of specific scheduling algorithms. These results cover optimality, feasibility, comparability, predictability, sustainability, and anomalies.

4.1. Optimality

As noted in Section 2.4 a scheduling algorithm is referred to as *optimal* if it can schedule all of the tasksets that can be scheduled by any other algorithm, that is, all of the feasible tasksets.

Horn [1974] gave an $O(N^3)$ algorithm (where N is the number of jobs) that is able to determine an optimal multiprocessor schedule for any arbitrary set of *completely determined* jobs where all of the arrival times and execution times are known a priori. This algorithm can be applied to a set of strictly periodic tasks, by considering all of the jobs in the hyperperiod; however, the $O(N^3)$ complexity means that it is only tractable for tasksets with a relatively short hyperperiod. This method is not applicable to sporadic tasksets where arrival times are not known in advance.

Hong and Leung [1988, 1992] proved that there is no optimal online scheduling algorithm for the case of an arbitrary collection of jobs that have more than one distinct deadline, and are scheduled on more than one processor. Hong and Leung [1988, 1992] showed that such an algorithm would require knowledge of future arrivals and execution times to avoid making decisions that lead to deadline misses; hence optimality in this case is impossible without clairvoyance. This result was extended by Dertouzos and Mok [1989], who showed that knowledge of arrival times is necessary for optimality, even if execution times are known.

Subsequently, Fisher [2007] proved that there is no optimal online algorithm for sporadic tasksets with constrained or arbitrary deadlines, by showing that such an algorithm would also require clairvoyance. Optimal algorithms are, however, known for periodic tasksets with implicit deadlines; see Section 6.3.

4.2. Feasibility

Horn [1974] observed the following necessary and sufficient condition for the feasibility of implicit-deadline periodic tasksets:

$$u_{sum} \leq m. \tag{6}$$

For constrained and arbitrary deadline tasksets, the above condition is necessary, but not sufficient. A tighter necessary condition given by Baruah and Fisher [2005] is

$$load(\tau) \leq m. \tag{7}$$

Baker and Cirinei [2006] improved upon this necessary feasibility condition by considering the modified processor load, that is, the processor load including task execution that must unavoidably take place within a time interval $[0, t)$, even though the task release time or deadline is not actually within the interval.

$$load^*(\tau) \leq m. \tag{8}$$

Baker and Cirinei [2006] showed that an upper bound on the modified processor load $load^*(\tau)$ can be found by considering a synchronous arrival sequence, with the modified processor load calculated from the modified processor demand bound function for each task:

$$h^*(t) = h(t) + \sum_{i=1}^{n} \max\left(0, t - \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) T_i - D_i + C_i\right). \tag{9}$$

Cucu and Goossens [2006] showed that the taskset hyperperiod $(0, H(\tau)]$ is a feasibility interval for implicit- and constrained-deadline synchronous periodic tasksets, scheduled by a deterministic and *memoryless*[1] algorithm. For any such algorithm, for example, global EDF, an exact schedulability test can be obtained by checking if the schedule generated misses any deadlines in $(0, H(\tau)]$. Further, an exact feasibility test for fixed-job-priority scheduling could in theory be achieved by checking the schedule for all $N!$ possible job priority orderings. It is not currently known if $(0, H(\tau)]$ is a feasibility interval for arbitrary deadline tasksets, under fixed-job-priority scheduling.

As far as we are aware, no exact feasibility test has yet been determined for sporadic tasksets scheduling by a fixed-job-priority algorithm.

Cucu and Goossens [2007] investigated the feasibility problem for fixed-task-priority algorithms. For this case, the above result for implicit- and constrained-deadline synchronous periodic tasksets holds as fixed-task-priority algorithms are both deterministic and memoryless. For arbitrary deadline periodic tasksets, Cucu and Goossens [2007] showed that the hyperperiod $(0, H(\tau)]$ is a feasibility interval provided that all previously released jobs are completed by $H(\tau)$. For asynchronous, periodic task systems, Cucu and Goossens [2007] showed that longer intervals are required to prove exact schedulability.

Cucu [2008] noted that, using the feasibility interval $(0, H(\tau)]$ and checking all $n!$ possible task priority orderings, it is in theory possible to determine exact feasibility for periodic tasksets scheduled using fixed-task priorities; however, this approach quickly becomes intractable as taskset cardinality increases.

As far as we are aware, no exact feasibility test or optimal priority ordering algorithm is known for sporadic tasksets scheduled using fixed-task priorities.

Fisher and Baruah [2007] devised a sufficient feasibility test for global scheduling of general task models. This test determines if a global scheduling algorithm exists that is able to schedule the taskset of interest. Unfortunately knowing that such an algorithm exists is of limited value without knowing what the algorithm is. The test, given by Equation (10) for sporadic tasksets with arbitrary deadlines, is sufficient as there are tasksets which it deems infeasible which are in fact feasible.

$$load(\tau) < \frac{m - (m-2)\delta_{\max}}{1 + \delta_{\max}}. \tag{10}$$

Fisher and Baruah [2007] showed that this feasibility test has a resource augmentation bound or speedup factor of $1/(\sqrt{2}-1) \approx 2.41$, meaning that any sporadic taskset that is feasible on m processors of speed $(\sqrt{2}-1)$ will be deemed feasible by the test on m processors of unit speed. Baruah and Fisher and [2008a] also derived a sufficient feasibility test for nonmigratory (i.e., partitioned) scheduling. This test states that there exists a partitioning of the tasks that is schedulable using EDF, which is an

[1]A *memoryless* algorithm makes scheduling decisions based only on the currently ready tasks, not on previous scheduling decisions.

Table II. Maximum Utilisation Bounds

Class	Maximum utilization bound
Global (job-level migration), dynamic priority	m
All other classes	$(m+1)/2$ (Andersson et al. [2001])

optimal uniprocessor scheduling algorithm, provided that

$$load(\tau) \leq \frac{1}{3}(m - (m-1)\delta_{\max}). \tag{11}$$

4.3. Comparability

In comparing the tasksets that can be scheduled by two different multiprocessor scheduling algorithms A and B, there are three possible outcomes.

(1) *Dominance*. Algorithm A is said to *dominate* algorithm B, if all of the tasksets that are schedulable according to algorithm B are also schedulable according to algorithm A, and tasksets exist that are schedulable according to A, but not according to B.
(2) *Equivalence*. Algorithms A and B are *equivalent*, if all of the tasksets that are schedulable according to algorithm B are also schedulable according to algorithm A, and vice versa.
(3) *Incomparable*. Algorithms A and B are *incomparable*, if there exist tasksets that are schedulable according to algorithm A, but not according to algorithm B and vice versa.

Carpenter et al. [2004] considered the relationships between the nine different classes of multiprocessor scheduling algorithm, comprising the combinations of the three migration-based and the three priority based categories described in Section 2.3. The key comparability results of Carpenter et al. [2004] are as follows.

—Global (i.e., job-level migration), dynamic priority scheduling *dominates* all other classes.
—All three classes with fixed task priorities (partitioned, task-level migration, and job-level migration) are *incomparable*. (Leung and Whitehead [1982] had previously shown that these partitioned and job-level migration classes are incomparable).
—All three partitioned classes (fixed-task priority, fixed-job priority, and dynamic priority) are incomparable with respect to all three task-level migration classes.

We note that, unlike uniprocessor scheduling where an optimal scheduling algorithm for periodic and sporadic tasksets exists in the fixed-job-priority class (i.e. EDF), in the case of multiprocessor scheduling, dynamic job priorities are essential for optimality.

The maximum possible utilization bounds, applicable to periodic tasksets with implicit deadlines, are given in Table II for algorithms in the various classes.

4.4. Predictability

Ha and Liu [1994] defined the concept of scheduling algorithm *predictability*. A scheduling algorithm is referred to as *predictable* if the response times of jobs cannot be increased by decreases in their execution times, with all other parameters remaining constant. Predictability is an important property, as in real systems task execution times are almost always variable up to some worst-case value. Ha and Liu [1994] proved that all priority-driven, that is, fixed-task priority or fixed-job priority, preemptive scheduling algorithms for multiprocessor systems are predictable. We note that, for any dynamic priority scheduling algorithm, it is necessary to prove predictability before the algorithm can be considered useful.

4.5. Sustainability

Baruah and Burns [2006] introduced the concept of *sustainability*. A scheduling algorithm is said to be *sustainable* with respect to a task model, if and only if schedulability of any taskset compliant with the model implies schedulability of the same taskset modified by (i) decreasing execution times, (ii) increasing periods or interarrival times, and (iii) increasing deadlines. Similarly, a schedulability test is referred to as *sustainable* if these changes cannot result in a taskset that was previously deemed schedulable by the test becoming unschedulable. We note that the modified taskset may not necessarily be deemed schedulable by the test. A schedulability test is referred to as *self-sustainable* [Baker and Baruah 2009] if such a modified taskset is always deemed schedulable by the test. We note that it is possible to devise sustainable sufficient schedulability tests for a scheduling algorithm that is unsustainable when an exact schedulability test is applied.

While EDF and fixed-priority scheduling are sustainable algorithms with respect to uniprocessor scheduling for both synchronous periodic and sporadic tasksets, the same is not true of global EDF and global fixed-task priority multiprocessor scheduling. This point is illustrated by the scheduling anomalies discussed in the next section. The sustainability of schedulability tests for global EDF has been investigated by Baker and Baruah [2009] and is discussed further in Section 6.1.

4.6. Anomalies

A scheduling *anomaly* occurs when a change in taskset parameters results in a counter-intuitive effect on schedulability. For example, increasing task periods while keeping all other parameters constant results in lower overall processor utilization, and so might reasonably be expected to improve schedulability; however, in some cases, this can result in the taskset becoming unschedulable. This effect is referred to as a *period anomaly* and is evidence of unsustainability.

4.6.1. Period and Execution Time Anomalies. In partitioned approaches to multiprocessor scheduling, *anomalies* exist in the task allocation/bin-packing algorithms used. These anomalies occur when a change in a parameter such as an increase in the period or a decrease in the worst-case execution time of a task results in a different allocation, which is then deemed to be unschedulable. Such anomalies are known to exist for EDF scheduling, in particular with FF (first-fit) and FFDU (first-fit decreasing utilization) allocation [Graham 1972]. These anomalies also exist for many fixed-task priority partitioning algorithms [Andersson 2003].

Andersson [2003] showed that global fixed-task priority scheduling of periodic tasksets using an exact schedulability test is also subject to period anomalies. In effect, the schedulability test is unsustainable with respect to increasing task periods.

Period anomalies are known to exist for global fixed-task priority scheduling of synchronous periodic tasksets, and for global optimal scheduling (full migration, dynamic priorities) of synchronous periodic tasksets. The interested reader is referred to Chapter 5 of Andersson [2003] for a set of illustrative examples.

4.6.2. Critical Instant Effect. Lauzac et al. [1998] showed that under global fixed-task priority scheduling, a task does not necessarily have its worst-case response time when released simultaneously with all higher-priority tasks. This happens because simultaneous release may not be the scenario that results in all processors being occupied by higher-priority tasks for the longest possible time during the interval over which the task of interest is active.

In multiprocessor scheduling, there are scenarios or patterns of task release which result in a longer response time for a low-priority task than that obtained by considering

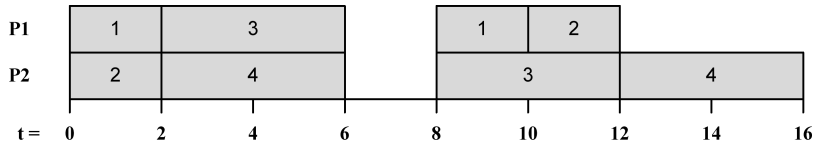

Fig. 1. Critical instant effect.

simultaneous release. These scenarios are characterized by a pattern of execution where, when the low-priority task executes, zero or only a few higher-priority tasks are executing on other processors, and, when other higher-priority tasks do execute, they do so together so that all processors are occupied and the low-priority task cannot execute.

The critical instant effect is a fundamental difference between global multiprocessor scheduling and partitioned/uniprocessor scheduling. In uniprocessor scheduling, synchronous release is known to represent the worst-case scenario for both periodic and sporadic tasksets.

The critical instant effect is illustrated by Figure 1. The task parameters (C_i,D_i,T_i) are as follows: $\tau_1(2,2,8)$, $\tau_2(2,2,10)$, $\tau_3(4,6,8)$, $\tau_4(4,7,8)$. The lowest-priority task τ_4 misses its deadline at time $t = 13$, despite meeting its deadline on the first invocation following simultaneous release of all four tasks. This happens because the higher-priority tasks occupy both processors for four time units in the interval [8, 15), whereas they only occupy both processors for two time units in the interval [0, 7).

Andersson [2003] observed that this effect has implications for priority assignment policies. In particular, the exact response time of a task is dependent on both the set of higher-priority tasks *and* their specific priority order. This implies that a greedy approach to priority assignment as used by Audsley's [1991, 2001] optimal priority assignment algorithm for the uniprocessor case is not applicable to the multiprocessor case, when schedulability analysis uses exact response times. Davis and Burns [2009] showed that this does not, however, rule out the use of Audsley's algorithm in conjunction with some sufficient schedulability tests.

The critical instant effect is also an issue in the analysis of global fixed-job priority scheduling. Baruah [2007] remarked that, "no finite collection of worst-case job arrival sequences has been identified for the global scheduling of sporadic task systems" (page 121). This problem remains one of the key open questions in the field today.

5. PARTITIONED SCHEDULING

In this section, we review the key research results in partitioned approaches to multiprocessor real-time scheduling.

Partitioned scheduling has the following advantages compared to global scheduling.

—If a task overruns its worst-case execution time budget, then it can only affect other tasks on the same processor.
—As each task only runs on a single processor, there is no penalty in terms of migration cost. For example, a job that is started on one processor, then preempted and resumed on another, must have its context restored on the second processor. This can result in additional communication loads and cache misses that would not occur in the partitioned/nonmigration case. This problem could be mitigated by allowing only task, as opposed to job-level migration, or by co-operative/nonpreemptive execution, although the latter could result in significant loss of schedulability due to long nonpreemptive sections.

Table III. Approximation Ratios

Algorithm	Approximation Ratio ($\Re_A$)	Reference
RMNF	2.67	Dhall and Liu [1978]
RMFF	2.33	Oh and Son [1993]
RMBF	2.33	Oh and Son [1993]
RRM-FF	2	Oh and Son [1995]
FFDUF	2	Davari and Dhall [1986]
RMST	$1/(1-u_{\max})$	Burchard et al. [1995]
RMGT	7/4	Burchard et al. [1995]
RMMatching	3/2	Rothvoss [2009]
EDF-FF	1.7	Garey and Johnson [1979]
EDF-BF	1.7	Garey and Johnson [1979]

—Partitioned approaches use a separate run-queue per processor rather than a single global queue. For large systems, the overheads of manipulating a single global queue can become excessive.

From a practical perspective, the main advantage of using a partitioning approach to multiprocessor scheduling is that, once an allocation of tasks to processors has been achieved, a wealth of real-time scheduling techniques and analyses for uniprocessor systems can be applied.

The following optimality results for uniprocessor scheduling had a strong influence on research into partitioned multiprocessor scheduling. Considering preemptive uniprocessor scheduling using fixed-task priorities, rate monotonic (RM) priority assignment is the optimal priority assignment policy for synchronous periodic or sporadic tasksets with implicit deadlines [Liu and Layland 1973]. Similarly, deadline monotonic (DM) priority assignment is optimal for such tasksets with constrained-deadlines [Leung and Whitehead 1982]. (We note that DM is not optimal for tasksets with arbitrary deadlines [Lehoczky 1990], or for asynchronous periodic tasksets; however, Audsley's [1991, 2001] priority assignment algorithm is known to be optimal in these cases). Considering preemptive uniprocessor scheduling using fixed-job priorities, EDF is the optimal scheduling algorithm for sporadic tasksets independent of the deadline constraints [Dertouzos 1974].

The main disadvantage of the partitioning approach to multiprocessor scheduling is that the task allocation problem is analogous to the bin packing problem and is known to be NP-Hard [Garey and Johnson 1979].

5.1. Implicit-Deadline Tasksets

Early research into partitioned multiprocessor scheduling by Dhall and Liu [1978], Davari and Dhall [1986], Oh and Son [1993, 1995], and Burchard et al. [1995] examined the use of EDF or fixed-priority scheduling using rate monotonic (RM) priority assignment, on each processor, combined with bin packing heuristics such as first fit (FF), next fit (NF), best fit (BF), and worst fit (WF), and task orderings such as decreasing utilisation (DU) for task allocation. In the following sections, these algorithms are referred to by their abbreviated names, for example RMBF means rate monotonic (fixed-priority) scheduling with best-fit task allocation.

5.1.1. Approximation Ratio. Table III gives the approximation ratio required for each of these algorithms for periodic tasksets with implicit deadlines. Recently, Rothvoss [2009] devised an $O(n^3)$ partitioning algorithm called RMMatching and showed that it has an approximation ratio of 3/2, improving upon the previous best approximation ratio of 7/4 for the fixed-task priority algorithm RMGT [Burchard et al. 1995]. (Note Table III is drawn from Zapata and Alvarez [2004] with some corrections and additions.)

While these approximation ratios enable a comparison to be made between the different algorithms, their practical use as a schedulability test is severely limited. This is because determining the minimum number of processors required by an optimal algorithm is, as noted above, an NP-hard problem. Also, the approximation ratio only holds as the number of processors required in the optimal case tends to infinity. Further, the utilization bounds that can be derived from these approximation ratios are pessimistic [Oh and Baker 1998].

5.1.2. Utilization Bounds. Andersson et al. [2001] showed that, for periodic tasksets with implicit deadlines, the largest worst-case utilization bound for any partitioning algorithm is

$$U_{OPT} = (m+1)/2. \tag{12}$$

Equation (12) holds because $m+1$ tasks with execution time $1+\varepsilon$ and a period of 2 cannot be scheduled on m processors regardless of the allocation algorithm used. The difficulties that partitioned scheduling has allocating large utilization tasks were recognized early on by the research community, leading to a significant thread of research during the 1990s aimed at determining utilization bounds as a function of $u_{\max}$, the highest utilization of any task in the taskset.

Burchard et al. [1995] provided utilization bounds for the RMST ("small tasks") algorithm, which attempts to place tasks with periods that are close to harmonics of each other on the same processor. This algorithm favors tasks with utilization $< 1/2$:

$$U_{RMST} = (m-2)(1-u_{\max}) + 1 - \ln 2. \tag{13}$$

Burchard et al. [1995] also provided utilization bounds for the RMGT ("general tasks") algorithm, which separates tasks into two groups depending on whether their utilization is above or below 1/3:

$$U_{RMGT} = \frac{1}{2}\left(m - \frac{5}{2}\ln 2 + \frac{1}{3}\right) \approx 0.5(m - 1.42). \tag{14}$$

Oh and Baker [1998] showed that RM-FFDU has a utilization bound given by

$$U_{RM-FFDU} = m(2^{1/2} - 1) \approx 0.41m. \tag{15}$$

Oh and Baker [1998] also showed that the utilization bound for any fixed-task priority partitioning algorithm is upper bounded by

$$U_{OPT(FTP)} < (m+1)/(1+2^{1/(m+1)}). \tag{16}$$

Lopez et al. [2003, 2004a, 2004b] subsequently generalized the above result for RM-FFDU, and also provided more complex bounds based on the number of tasks and the value of $u_{\max}$ for RMBF, RMFF, and RMWF.

Andersson and Jonsson [2003] showed that the RBOUND-MP-NFR algorithm has a utilization bound of

$$U_{RBOUND-MP-NFR} = m/2. \tag{17}$$

This result shows that a fixed-task priority partitioning algorithm exists that is an optimal partitioning approach in the limited sense that its utilization bound is the maximum possible for any partitioning algorithm. We note that this does not mean that it is an optimal partitioning algorithm in the sense that it can schedule any taskset that is schedulable according to any other partitioning algorithm.

Lopez et al. [2000] showed that, using EDF, the lowest utilization bound for any *reasonable*[2] allocation algorithm is given by

$$L_{RA} = m - (m-1)u_{\max} \tag{18}$$

and that the highest utilization bound of any reasonable allocation algorithm is

$$H_{RA} = \frac{(\lfloor 1/u_{\max} \rfloor m + 1)}{(\lfloor 1/u_{\max} \rfloor + 1)}. \tag{19}$$

(Note, these limits assume that $n > m/(\lfloor 1/u_{\max} \rfloor)$.)

Lopez et al. [2000] showed that all reasonable allocation algorithms that order tasks by decreasing utilization achieve the higher limit, as do EDF-BF and EDF-FF. Further, EDF-WF, but not EDF-WFDU, achieves the lower limit. When $u_{\max} = 1$, the limit given by Equation (19), becomes the same as Equation (12); hence EDF-FF and EDF-BF are also "optimal" partitioning approaches in the limited sense that their utilization bounds are as large as that, of any partitioning algorithm. We note that, for applications with "small" tasks, RMST and EDF-FF provide reasonably high utilization bounds. For example, assuming $m = 10$ and $u_{\max} = 0.25$, the utilization bounds for RMST and EDF-FF are 63% and 82%, respectively.

5.2. Constrained and Arbitrary Deadline Tasksets

Baruah and Fisher [2005] showed that EDF-FFD (decreasing density) is able to schedule any arbitrary-deadline sporadic taskset provided that

$$\delta_{sum} \leq \begin{cases} m - (m-1)\delta_{\max}, & \delta_{\max} \leq 1/2, \\ m/2 + \delta_{\max}, & \delta_{\max} \geq 1/2. \end{cases} \tag{20}$$

However, the resource augmentation factor for EDF-FFD is *not* finite.

Baruah and Fisher [2005, 2006, 2007] also developed an algorithm called EDF-FFID based on ordering tasks by increasing relative deadline, and using a sufficient test based on a linear upper bound for the processor demand bound function to determine schedulability. They showed that EDF-FFID is able to schedule any sporadic taskset with constrained deadlines provided that

$$m \geq \left(\frac{2load(\tau) - \delta_{\max}}{1 - \delta_{\max}} \right). \tag{21}$$

For tasksets with arbitrary deadlines, the test becomes

$$m \geq \frac{load(\tau) - \delta_{\max}}{1 - \delta_{\max}} + \frac{u_{sum} - u_{\max}}{1 - u_{\max}}. \tag{22}$$

The resource augmentation or speedup factor required by this algorithm is

- $(2 - 1/m)$ for tasksets with implicit deadlines;
- $(3 - 1/m)$ for tasksets with constrained deadlines;
- $(4 - 2/m)$ for tasksets with arbitrary deadlines.

Fisher et al. [2006] applied a similar approach to the problem of partitioning using fixed-task priority scheduling using deadline monotonic priority assignment. The algorithm FFB-FFD (from the author's surnames) is based on ordering tasks by decreasing relative deadline, and using a sufficient test based on a linear upper bound on the processor request bound function to determine schedulability. They showed that FFB-FFD

[2]A reasonable allocation algorithm is one that only fails to allocate a task once there is no processor on which the task will fit.

is able to schedule any sporadic taskset with constrained deadlines provided that

$$m \geq \frac{load(\tau) + u_{sum} - \delta_{\max}}{1 - \delta_{\max}}. \tag{23}$$

For tasksets with arbitrary deadlines, the test becomes

$$m \geq \frac{load(\tau) + u_{sum} - \delta_{\max}}{1 - \delta_{\max}} + \frac{u_{sum} - u_{\max}}{1 - u_{\max}}. \tag{24}$$

Fisher et al. [2006] showed that the resource augmentation or speedup factor required by this algorithm is

$(3 - 1/m)$ for tasksets with constrained deadlines;
$(4 - 2/m)$ for tasksets with arbitrary deadlines.

6. GLOBAL SCHEDULING

In this section we outline the key research results in global multiprocessor scheduling where tasks are permitted to migrate from one processor to another.

Global scheduling has the following advantages compared to partitioned scheduling.

—There are typically fewer context switches/preemptions when global scheduling is used. This is because the scheduler will only preempt a task when there are no processors idle [Andersson and Jonsson 2000a].
—Spare capacity created when a task executes for less than its worst-case execution time can be utilized by all other tasks, not just those on the same processor.
—If a task overruns its worst-case execution time budget, then there is arguably a lower probability of deadline failure as worst-case behavior of the entire system, with all tasks taking worst-case execution times, worst-case phasing occurring, etc., is less likely across multiple processors than it is on a single processor.
—Global scheduling is more appropriate for open systems, as there is no need to run load balancing/task allocation algorithms when the set of tasks changes.

The majority of the research into global real-time scheduling has focused on approaches that permit job-level migration, where a job may be preempted on one processor and resumed on another. In the descriptions that follow, job-level migration should be assumed unless task-level migration, where each job executes on a single processor but jobs of the same task may execute on different processors, is explicitly stated.

The seminal work of Dhall and Liu [1978] considered global scheduling of periodic tasksets with implicit deadlines on m processors. They showed that the utilization bound for global EDF scheduling is $1 + \varepsilon$, for arbitrarily small ε. This occurs when there are m tasks with short periods/deadlines and infinitesimal utilization, and one task with a longer period/deadline and a utilization that approaches 1. This Dhall effect led to a general view that global approaches to multiprocessor scheduling are inferior to partitioned approaches. As a result, throughout the 1980s and early 1990s, the majority of research into multiprocessor real-time scheduling focused on partitioned approaches, as described in the previous section.

Phillips et al. [1997] showed that augmenting a system by increasing processor speed is more effective than augmenting a system by adding processors. They showed that the resource augmentation or speedup factor required for global EDF is at most $(2 - 1/m)$. This result also applies to global least laxity first (LLF), which can schedule any taskset that is schedulable by global EDF. The resource augmentation results of Phillips et al. [1997], along with research by Funk et al. [2001] into uniform multiprocessor scheduling, led to the observation that for the Dhall effect to occur at least one task is needed with very high utilization. This observation was exploited in much of the

subsequent research to provide utilization bounds that are dependent on the maximum task utilization $u_{\max}$.

6.1. Global Fixed-Job-Priority Scheduling

6.1.1. Implicit Deadline Tasksets. Andersson et al. [2001] considered utilization bounds for periodic tasksets with implicit deadlines. They showed that the maximum utilization bound for any global fixed job priority algorithm is

$$U_{OPT} = (m+1)/2. \tag{25}$$

Srinvasan and Baruah [2002] proposed the EDF-US[ς] algorithm that gives the highest priority to tasks with utilization greater than the threshold ς, with ties broken arbitrarily. Setting the threshold to $m/(2m-1)$ results in a utilization bound that is independent of $u_{\max}$:

$$U_{EDF-US[m/(2m-1)]} = m^2/(2m-1). \tag{26}$$

Goossens et al. [2003] derived a utilization bound for global EDF applicable to periodic tasksets with implicit deadlines and showed that this bound is tight:

$$U_{EDF} = m - (m-1)u_{\max}. \tag{27}$$

Baruah and Carpenter [2003] showed that this same utilization bound applies to global EDF scheduling, assuming task-level migration. Goossens et al. [2003] also proposed an algorithm called EDF(k) that assigns the highest priority to the k tasks with the highest utilization. They showed that a sufficient schedulability condition for EDF(k) is

$$m \geq (k-1) + \left\lceil \frac{u_{sum} - u_k}{1 - u_k} \right\rceil, \tag{28}$$

where u_k is the utilization of the kth task, in order of decreasing utilization.

Baker [2005] (see also Baker and Baruah [2007a]) showed that setting the threshold used in EDF-US[ς] to 1/2 results in the following utilization bound which is the maximum possible bound for this class of algorithm [Andersson et al. 2001]:

$$U_{EDF-US[1/2]} = (m+1)/2. \tag{29}$$

Baker [2005] also proposed a variant of EDF(k) called *EDF*($k_{\min}$), where $k_{\min}$ is the minimum value of k for which the sufficient test in Equation (29) holds. Baker [2005] showed that the utilization bound for EDF($k_{\min}$) is also

$$U_{EDF[k_{\min}]} = (m+1)/2. \tag{30}$$

Again, this is the maximum possible utilization bound for this class of scheduling algorithm. However, EDF($k_{\min}$) dominates EDF-US[1/2] in terms of the tasksets that it can schedule.

6.1.2. Constrained and Arbitrary-Deadline Tasksets. The proof of the utilization bound given in Equation (27) was extended by Bertogna et al. [2005b] to the case of sporadic tasksets with constrained deadlines and by Baker and Baruah [2007b] to the arbitrary-deadline case, giving the following sufficient schedulability test based on task density:

$$\delta_{sum} \leq m - (m-1)\delta_{\max}. \tag{31}$$

Bertogna [2007] also adapted the utilization separation approach of EDF-US to the case of sporadic tasksets with constrained and arbitrary deadlines, forming the EDF-DS[ς] algorithm. This algorithm gives the highest priority to tasks with density greater

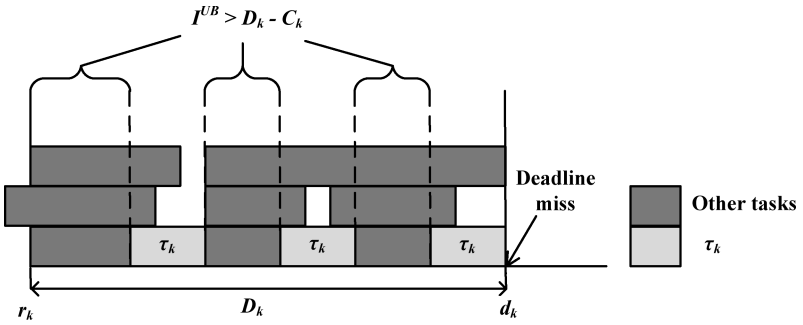

Fig. 2. Problem window.

than the threshold ς. Bertogna [2007] showed that a sporadic taskset is schedulable according to EDF-DS[1/2] provided that

$$\delta_{sum} \leq (m+1)/2. \tag{32}$$

Baker [2003] developed a general strategy for determining the schedulability of sporadic tasksets. The outline of this basic strategy is as follows.

(1) Consider an interval, referred to as the *problem window*, at the end of which a deadline is missed (see Figure 2), for example the interval $[r_k, d_k]$ from the release to the deadline of some job of task τ_k.
(2) Establish a condition *necessary* for the job to miss its deadline, for example, all m processors execute other jobs for more than $D_k - C_k$ during the interval.
(3) Derive an upper bound I^{UB} on the maximum interference in the interval due to jobs of other tasks, including jobs released in the interval and so called *carry-in* jobs that have not completed execution before the start of the interval.
(4) Form a necessary unschedulability test, in the form of an inequality between I^{UB} and the amount of execution necessary for a deadline to be missed.
(5) Negate this inequality to form a sufficient schedulability test.

The idea presented by Baker [2003] is that, if the job of task τ_k misses its deadline, then the load in the interval must be at least $m(1-\delta_k)+\delta_k$. In order to improve the estimate of execution time carried in, Baker [2003] extended the interval back as far as possible before the release of the job, such that the load remained just greater than $m(1-\delta_k)+\delta_k$. This gives the following sufficient schedulability test: a constrained-deadline taskset is schedulable under preemptive global EDF scheduling if for every task τ_k.

$$\sum_{\forall i} \min(1, \beta_i) < m(1-\delta_k)+\delta_k, \tag{33}$$

where β_i is an upper bound on the processor load due to task τ_i for any problem window relating to τ_k. See Lemma 11 in Baker [2003] for a definition of β_i. Baker [2005] extended this approach to sporadic tasksets with arbitrary deadlines. We note that the complexity of Baker's test is $O(n^3)$ in the number of tasks. The basic strategy proposed by Baker [2003] is a seminal result which has been built upon by a significant thread of subsequent research.

Bertogna et al. [2005b] showed that the test proposed by Baker [2005] (Equation (33)) does not dominate the extended version of test proposed by Goossens et al. [2003] (Equation (31)). In fact, the test given by Equation (33) performs relatively poorly when tasks with high individual utilizations are considered. Bertogna et al. [2005b] proposed an alternative sufficient test based on the strategy of Baker, but using some simple

observations to limit the amount of interference counted as falling in the problem window. This sufficient test can be summarized as follows: a constrained-deadline taskset is schedulable under preemptive global EDF scheduling if, for every task τ_k, one of the following holds:

$$\sum_{\forall i \neq k} \min(\beta_k(i), 1-\delta_k) < m(1-\delta_k) \quad \text{or} \quad \sum_{\forall i \neq k} \min(\beta_k(i), 1-\delta_k) = m(1-\delta_k)$$
$$\text{and} \quad \exists i \neq k : 0 < \beta_k(i) \leq 1-\delta_k, \tag{34}$$

where

$$\beta_k(i) = \frac{N_i C_i + \min(C_i, (D_k - N_i T_i)_0)}{D_k}. \tag{35}$$

The complexity of this test is $O(n^2)$ in the number of tasks.

We note that the schedulability tests given by Equations (33) and (34) become pessimistic when the number of tasks is much greater than the number of processors ($n \gg m$). This happens because every task is counted as contributing some carry-in interference. Further, these tests tend to perform poorly on tasksets where the parameters of different tasks are of different orders of magnitude.

Baruah [2007] derived a sufficient schedulability test for global EDF scheduling of sporadic tasksets with constrained deadlines. This test uses the same basic approach as Baker [2003] but extends the interval during which task execution is considered back to some point in time t_0 at which at least one of the m processors is idle. In this way, the test limits the number of tasks that are counted as causing carry-in interference to $m-1$. For each task, the schedulability test presented by Baruah [2007] checks values of A_k representing the time interval between t_0 and the arrival of the first job of task τ_k to miss its deadline. The range of values of A_k to be checked is constrained by the following upper bound:

$$A_k \leq \frac{C_\Sigma - D_k(m - u_{sum}) + \sum (T_i - D_i)u_i + mC_k}{m - u_{sum}}, \tag{36}$$

where C_Σ is the sum of the $m-1$ largest task execution times.

Within this range of possible values for A_k, only those values where the processor demand bound function $h(A_k + D_k)$ changes need to be checked, making the test pseudopolynomial in complexity.

A constrained-deadline taskset is schedulable under preemptive global EDF scheduling if, for every task τ_k, the following holds for all values of A_k:

$$\sum_{\forall i} I'_k(i) + I^{\in}_k < m(A_k + D_k - C_k), \tag{37}$$

where

$$I^{\in}_k = \sum_{\forall i (m-1) l \arg est} (I''_k(i) - I'_k(i)),$$

$$I'_k = \begin{cases} \min(h_i(A_k + D_k), A_k + D_k - C_k), & i \neq k, \\ \min(h_i(A_k + D_k) - C_k, A_k), & i = k, \end{cases}$$

$$I''_k = \begin{cases} \min \left(\left\lfloor \dfrac{A_k + D_k}{T_i} \right\rfloor C_i + \min(C_i, (A_k + D_k) \bmod T_i) \right), & i \neq k, \\ A_k + D_k - C_k \\ \min \left(\left\lfloor \dfrac{A_k + D_k}{T_i} \right\rfloor C_i + \min(C_i, (A_k + D_k) \bmod T_i - C_k \right), & i = k. \\ A_k \end{cases}$$

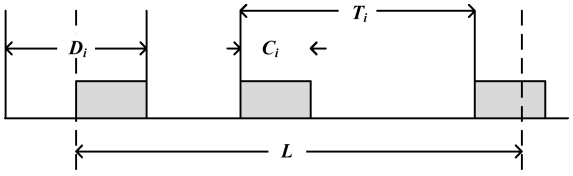

Fig. 3. Densest packing for work-conserving algorithms.

Notably, the above test reverts to the processor load-based test ($load(\tau) \leq 1$) for uniprocessor systems [Baruah et al. 1990a, 1990b] when $m = 1$.

Building on their previous work Baker and Baruah [2008] derived a further sufficient test for global EDF scheduling of sporadic tasksets with constrained deadlines, which also limits the number of tasks that are counted as causing carry-in interference. This processor load based test is given by Equation (38), where $\mu = m - (m-1)\delta_{\max}$:

$$load(\tau) \leq \mu - (\lceil \mu \rceil - 1)\delta_{\max}. \tag{38}$$

Baker and Baruah [2008] showed that the sufficient test given by Equation (38), combined with global EDF scheduling, has a resource augmentation or speedup factor of

$$f = \frac{2}{3 - \sqrt{5}} \approx 2.62. \tag{39}$$

This speedup factor is sufficient to compensate for *both* the nonoptimality of global EDF and the sufficiency of the test.

Latter in the same year, Baruah and Baker [2008] extended the results given in Baker and Baruah [2008] to sporadic tasksets with arbitrary deadlines, showing that Equation (38) still applies. Baruah and Baker [2009] also extended their results to the case where jobs of an arbitrary deadline task may execute in parallel on different processors. They showed that, unlike the partitioned case (see Section 5.2), there appears to be no performance penalty for permitting arbitrary deadlines. However, the analysis provided is only sufficient, and the speedup factor derived is an upper bound, so it is possible that the lack of a penalty could be an artifact of the analysis used.

Bertogna et al. [2008] presented a schedulability test for sporadic tasksets with constrained deadlines that is valid for any work conserving algorithm. This schedulability test is based on a consideration of the densest possible packing of interfering jobs in the problem window; see Figure 3.

Bertogna et al. [2008] showed that $W_i(L)$ is an upper bound on the workload of task τ_i in an interval of length L.

$$W_i(L) = N_i(L)C_i + \min(C_i, L + D_i - C_i - N_i(L)T_i), \tag{40}$$

where $N_i(L)$ is the maximum number of jobs of task τ_i that contribute all of their execution time in the interval.

$$N_i(L) = \left\lfloor \frac{L + D_i - C_i}{T_i} \right\rfloor. \tag{41}$$

A taskset is therefore schedulable with any work-conserving global scheduling algorithm if for each task τ_k

$$\sum_{i \neq k} \min(W_i(D_k), D_k - C_k + 1) < m(D_k - C_k + 1). \tag{42}$$

Bertogna et al. [2008] extended this test to the specific cases of global EDF, and global FP (fixed-task priority) scheduling (see Section 6.2.2). Under global EDF, they showed

that a taskset is schedulable provided that for each task τ_k the following holds:

$$\sum_{i \neq k} \min\left(\Lambda_k^i(D_k), D_k - C_k + 1\right) < m(D_k - C_k + 1), \tag{43}$$

where

$$\Lambda_k^i = \left\lfloor \frac{D_k}{T_i} \right\rfloor C_i + \min\left(C_i, D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i\right).$$

Bertogna et al. [2008] further extended their approach via an iterative schedulability test that calculates the slack for each task, and then uses this value to limit the amount of carry-in interference and hence calculate a new value for the task slack. This approach is also applicable to any work-conserving algorithm and was also specialized for global EDF and global FP scheduling. They showed that the iterative test for global EDF admits nearly as many randomly generated tasksets as the sufficient feasibility test of Fisher and Baruah [2007]; see Section 4.2, Equation (10). This iterative test has a complexity that is pseudopolynomial: $O(n^3 D_{\max}^2)$.

Baruah and Fisher [2008a] derived the following sufficient test for jobs of sporadic tasks scheduled by global EDF. Note, $load(\tau, j)$ is the processor load due to all jobs with higher priority than job j, and K is the largest ratio of task deadlines. As K may potentially take any value, this test does not have a finite resource augmentation factor.

$$load(\tau, j) \leq \frac{1}{1+K}(m - (m-1)C_j/D_j). \tag{44}$$

Baker and Baruah [2009] showed that the sufficient schedulability tests for global EDF given in Baker [2005], Baker and Baruah [2008], and Bertogna et al. [2005b] (Equations (33), (34) and (38)) are unsustainable with respect to increases in relative deadline, and the test given by Baker and Baruah [2008] (Equation (38)) is unsustainable with respect to decreases in worst-case execution times. That is, increases in relative deadlines (decreases in worst-case execution times) can result in a taskset being deemed unschedulable when it was previously deemed schedulable by the test. Baruah and Baker [2009] improved the sufficient test given by Equation (38), making it sustainable. The improved schedulability test is as follows:

$$load(\tau) \leq \max((\mu - (\lceil \mu \rceil - 1)\delta_{\max}), ((\lceil \mu \rceil - 1) - (\lceil \mu \rceil - 2)\delta_{\max})), \tag{45}$$

where $\mu = m - (m-1)\delta_{\max}$.

Bonifaci et al. [2008] derived a sufficient schedulability test for global EDF scheduling of sporadic tasksets with arbitrary deadlines which has a speedup factor of $(2 - 1/m + \varepsilon)$ for arbitrarily small ε. Recall that Phillips et al. [1997] showed that global EDF requires m processors of speed $(2 - 1/m)$ in order to schedule all tasksets that are feasible on m processors of unit speed. The schedulability test introduced by Bonifaci et al. [2008] and extended by Baruah et al. [2009] therefore has the property that there are no tasksets that are feasible on m processors of unit speed that are not deemed to be schedulable by the test under global EDF on m processors of speed $(2 - 1/m)$. In this sense, the test is speedup optimal, as no schedulability test exists for global EDF that requires a smaller speedup factor.

6.2. Global Fixed-Task-Priority Scheduling

This section outlines research into global fixed-task priority scheduling. For conciseness we use the following abbreviated descriptions for various scheduling algorithms: global FP scheduling (meaning global fixed-task priority scheduling), global RM scheduling (meaning global FP scheduling using rate monotonic priority ordering), and global DM scheduling (meaning global FP scheduling using deadline monotonic priority ordering).

6.2.1. Implicit Deadline Tasksets. As well as global EDF scheduling, discussed in Section 6.1, the seminal work of Dhall and Liu [1978] also considered global scheduling of periodic tasksets with implicit deadlines on m processors. They showed that the utilization bound for global RM scheduling is $1+\varepsilon$, for arbitrarily small ε. This occurs when there are m tasks with short periods/deadlines and infinitesimally small utilization, and one task with a longer period/deadline and utilization that approaches 1.

Andersson and Jonsson [2000a] designed the TkC priority assignment policy to circumvent the Dhall effect. TkC assigns priorities based on a task's period T_i minus k times its worst-case execution time C_i, where k is a real value computed on the basis of the number of processors.

$$k = \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{2m}. \tag{46}$$

Via an empirical investigation, Andersson and Jonsson [2000a] showed that TkC is an effective priority assignment policy for periodic tasksets with implicit deadlines.

Andersson et al. [2001] showed that any periodic taskset with implicit deadlines can be scheduled using global RM scheduling provided that

$$u_{\max} \leq m/(3m-2) \text{ and } u_{sum} \leq m^2/(3m-1). \tag{47}$$

This result, albeit in a weaker form, also appeared in Baruah and Goossens [2003]:

$$u_{\max} \leq 1/3 \text{ and } u_{sum} \leq m/3. \tag{48}$$

Andersson et al. [2001] also proposed the RM-US[ς] algorithm that gives the highest priority to tasks with utilization greater than the threshold ς (with ties broken arbitrarily), and otherwise assigns priorities in RM order. Andersson et al. [2001] showed that RM-US[$m/(3m-2)$] has a utilization bound of

$$U_{RM-US[m/(3m-2)]} = m^2/(3m-1). \tag{49}$$

Lundberg [2002] showed that setting the threshold used in RM-US[ς] to 0.375 results in the following utilization bound which is the maximum possible bound for this algorithm:

$$U_{RM-US[0.375]} \approx 0.375m. \tag{50}$$

Lundberg and Lennerstad [2007] gave a utilization bound of $3-\sqrt{7} \approx 0.354$ for RM-US[ς], for aperiodic tasks where the utilization threshold is based on the maximum synthetic utilization (see Lundberg and Lennerstad [2007] for a definition). Further, they gave a formula for the utilization bound as a function of m.

Andersson and Jonsson [2003] showed that, for periodic tasksets with implicit deadlines, the maximum utilization bound for any global fixed-task priority scheduling algorithm where priorities are defined as a scale-invariant function of task periods and worst-case execution times is

$$U_{OPT} \leq (\sqrt{2}-1)m \approx 0.41m. \tag{51}$$

Bertogna et al. [2005a] tightened the bound for global RM scheduling to

$$u_{sum} \leq \frac{m}{2}(1 - u_{\max}) + u_{\max}. \tag{52}$$

Andersson [2008] proposed a "slack monotonic" algorithm, where priorities are ordered according to the slack of each task given by $T_i - C_i$. This algorithm, called *SM-US*, otherwise works in the same way as RM-US. Andersson [2008] showed that

SM-US[$2/(3+\sqrt{5})$] has the following utilization bound of for sporadic tasksets with implicit deadlines:

$$U_{SM-US[2/(3+\sqrt{5})]} = 2/(3+\sqrt{5})m \approx 0.382m. \tag{53}$$

Essentially the same result was obtained independently by Lundberg and Lennerstad [2008] for aperiodic tasks where the utilization threshold is based on the maximum synthetic utilization. They also gave a formula for the utilization bound based on the number of processors, and showed that SM-US outperforms RM-US for $m > 2$.

6.2.2. Constrained and Arbitrary Deadline Tasksets. Andersson and Jonsson [2000b] gave a simple, but pessimistic, response time upper bound applicable to sporadic tasksets with constrained deadlines scheduled using fixed priorities. This response time upper bound effectively assumes that the execution time of carried-in and carried-out jobs in an interval is equal to the entire worst-case execution time of the task.

$$R_k^{ub} \leftarrow C_k + \frac{1}{m}\sum_{i<k}\left(\left\lceil \frac{R_k^{ub}}{T_i}\right\rceil C_i + C_i\right). \tag{54}$$

Baker [2003, 2006a] applied the same general strategy described in Section 6.1.2 for global EDF to global FP scheduling of sporadic tasksets with constrained deadlines.

Bertogna et al. [2005a] proved the following density bound for global DM scheduling of sporadic tasksets with constrained deadlines:

$$\delta_{sum} \le \frac{m}{2}(1-\delta_{\max}) + \delta_{\max}. \tag{55}$$

Bertogna et al. [2005a] used the above result as the basis for a density-based test for the hybrid DM-DS[ς] algorithm. This algorithm gives the highest priority to at most $m-1$ tasks with density greater than the threshold ς, and otherwise assigns priorities in deadline monotonic priority order. Under DM-DS[ς] a taskset is schedulable provided that

$$\delta_{sum} \le \psi\varsigma + \begin{cases} \delta^{(m)} + \ln\dfrac{2}{1+\delta^{(m)}}, & \psi = m-1, \\ \varsigma + \dfrac{m-\psi}{2}(1-\varsigma), & \psi < m-1, \end{cases} \tag{56}$$

where ψ is the number of "privileged" tasks with density higher than the threshold, and $\delta^{(m)}$ is the density of the mth highest density task.

Bertogna et al. [2005a] proved the following sufficient test for DM-DS[1/3]:

$$\delta_{sum} \le \frac{m+1}{3}. \tag{57}$$

Bertogna et al. [2005a] also proposed the following alternative sufficient test based on the strategy of Baker [2003], but using some simple observations to limit the amount of interference counted as falling in the problem window.

A constrained-deadline taskset is schedulable under preemptive global DM scheduling if, for every task τ_k, one of the following holds:

$$\sum_{i<k}\min(\beta_k(i), 1-\delta_k) < m(1-\delta_k) \quad \text{or} \quad \sum_{i<k}\min(\beta_k(i), 1-\delta_k) = m(1-\delta_k)$$
$$\text{and} \quad \exists i \ne k : 0 < \beta_k(i) \le 1-\delta_k, \tag{58}$$

where

$$\beta_k(i) = \frac{N_{i,k}C_i + \min(C_i, (D_k - N_{i,k}T_i + D_i - C_i)_0)}{D_k} \quad \text{and}$$
$$N_{i,k} = \left\lfloor \frac{D_k - C_i}{T_i} \right\rfloor + 1.$$

Fisher and Baruah [2006], derived a sufficient test for global DM scheduling of sporadic tasksets with arbitrary deadlines, under the assumption that *intratask parallelism*, where jobs of the same task can execute in parallel on different processors, is permitted, while *interjob parallelism* is not. This sufficient test for each task τ_k is as follows:

$$load(\tau, k) \leq \frac{1}{1 + 2(\max_{j \in hp(k)}(D_j/D_k)}(m - (m-1)C_k/D_k), \tag{59}$$

where $load(\tau, k)$ is the processor load due to all tasks with priority higher than or equal to k. Note, due to the use of deadline monotonic priority ordering, the minimum value for the fractional term is 1/3.

Baruah [2007] derived an alternative sufficient test for global DM scheduling of sporadic tasksets with constrained deadlines using a similar approach to Fisher and Baruah [2006], but limiting the amount of carry-in execution in a different way. This sufficient test for each task τ_k is as follows:

$$load(\tau, k) \leq \frac{1}{2}(m - (m-1)C_k/D_k - C_\Sigma(k)/D_k), \tag{60}$$

where $C_\Sigma(k)$ is the sum of the m largest worst-case execution times of tasks of priority k or higher. The above test can be weakened to

$$load(k) \leq \frac{1}{2}(m - (m-1)C_k/D_k)(1 - \delta_{\max}(k)), \tag{61}$$

where $\delta_{\max}(k)$ is the maximum density of any of the k highest-priority tasks. Baruah [2007] showed that this schedulability test has a resource augmentation or speedup factor for large m of $(2 + \sqrt{3}) \approx 3.73$, which compensates for both the nonoptimality of global DM scheduling and the sufficiency of the test.

Baruah and Fisher [2008b] showed that the result derived in Fisher and Barauh [2006], Equation (59), also applies to systems where intratask parallelism is not permitted. They showed that DM is the optimal priority assignment policy with respect to this schedulability test and that the test has a resource augmentation or speedup factor of $(4 - 1/m)$.

Bertogna et al. [2008] specialized their sufficient schedulability test for any work-conserving algorithm, (see Section 6.1.2 Equation (42)), to global FP scheduling. They showed that a sporadic taskset with constrained deadlines is schedulable under global FP scheduling if for each task τ_k

$$\sum_{i<k} \min(W_i(D_k), D_k - C_k + 1) < m(D_k - C_k + 1), \tag{62}$$

where $W_i(L)$ is the bound on the workload of task τ_i in an interval of length L, given by Equation (40).

Bertogna et al. [2008] extended their approach via an iterative schedulability test that calculates the slack for each task, and then uses this value to limit the amount of carry-in interference and hence calculate a new value for the task slack.

Further, Bertogna and Cirinei [2007] showed how this approach could be adapted to provide response time analysis for multiprocessor systems, by iteratively computing an upper bound on the response time of each task, while using the response times of higher-priority tasks to limit the carry-in interference from those tasks. This analysis can be expressed as in the following fixed-point iteration:

$$R_k^{ub} \leftarrow C_k + \frac{1}{m} \sum_{i<k} \min\left(I_i^{CI}\left(R_k^{UB}\right)\right), \tag{63}$$

where, assuming that τ_k is schedulable, $I_i^{CI}(R_k^{UB})$ is an upper bound on the interference due to task τ_i within the worst-case response time of τ_k, given by

$$I_i^{CI}\left(R_k^{UB}\right) = \min\left(W_i^{CI}(R_k^{UB}), R_k^{UB} - C_k + 1\right), \tag{64}$$

where $W_i^{CI}(L)$ is a bound on the workload of task τ_i in an interval of length L, given by

$$W_i^{CI}(L) = N_i^{CI}(L)C_i + \min\left(C_i, L + R_i^{ub} - C_i - N_i^{CI}(L)T_i\right), \tag{65}$$

and $N_i^{CI}(L)$ is the maximum number of jobs of task τ_i that contribute all of their execution time in the interval:

$$N_i^{CI}(L) = \left\lfloor \frac{L + R_i^{ub} - C_i}{T_i} \right\rfloor. \tag{66}$$

Guan et al. [2009] extended the response time analysis of Bertogna and Cirinei [2007], limiting the amount of "carry-in" interference using ideas from Baruah [2007]. An important observation following from the analysis of Guan et al. [2009] concerns the pattern of task execution that results in the worst-case response time for a job of task τ_k under global FP scheduling: the worst-case response time for a job of task τ_k occurs when that job is released at some time t when all m processors are busy executing higher-priority tasks, and during the preceding time interval $[t - \varepsilon, t)$ (for some arbitrary value of ε) at least one processor was not occupied by a higher priority task. Guan et al. [2009] showed that, if task τ_i does not have a carry-in job, then the interference is given by

$$I_i^{NC}\left(R_k^{UB}\right) = \min\left(W_i^{NC}\left(R_k^{UB}\right), R_k^{UB} - C_k + 1\right), \tag{67}$$

where

$$W_i^{NC}(L) = N_i^{NC}(L)C_i + \min\left(C_i, L - N_i^{NC}(L)T_i\right) \tag{68}$$

and

$$N_i^{NC}(L) = \left\lfloor \frac{L}{T_i} \right\rfloor. \tag{69}$$

The difference between the two interference terms (Equations (64) and (67)) is given by

$$I_i^{DIFF}\left(R_k^{UB}\right) = I_i^{CI}\left(R_k^{UB}\right) - I_i^{NC}\left(R_k^{UB}\right). \tag{70}$$

Using this result, Guan et al. [2009] improved upon the response time test of Bertogna and Cirinei [2007] as follows:

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}\left(R_k^{UB}\right) + \sum_{i \in Max(k, m-1)} I_i^{DIFF}\left(R_k^{UB}\right) \right) \right\rfloor, \tag{71}$$

where $Max(k, m-1)$ is the subset of tasks with higher priorities than τ_k, with the $m-1$ largest values of $I_i^{DIFF}(R_k^{UB})$. The improved response time test of Guan et al.

[2009] (Equation (71)) dominates the response time test of Bertogna and Cirinei [2007] (Equation (63)), which in turn dominates the deadline-based test of Bertogna et al. [2008]. Guan et al. [2009] also extended their response time test to tasksets with arbitrary deadlines.

Davis and Burns [2009] showed that Audsley's [1991, 2001] optimal priority assignment algorithm is applicable to some sufficient tests for global FP scheduling, including those of Andersson and Jonsson [2000b] (Equation (54)) and Bertogna et al. [2008] (Equation (62)). Davis and Burns [2009] also extended the TkC priority assignment policy to sporadic tasksets to form the "DkC" priority assignment, which orders task priorities based on their deadlines less some constant (given by Equation (46)) multiplied by their worst-case execution times.

6.3. Global Dynamic Priority Scheduling

In this section, we outline research into global dynamic priority scheduling algorithms. A number of these algorithms are known to be optimal for periodic tasksets with implicit deadlines (Pfair and its variants PD, PD^2, ERFair, BF, and also SA [Khemka and Shyamasundar 1997], and LLREF). However, it is known that there are no optimal online (nonclairvoyant) algorithms for the preemptive scheduling of sporadic tasksets on multiprocessors [Fisher 2007].

Global dynamic priority algorithms dominate algorithms in all other classes; however, their practical use can be problematic due to the potentially excessive overheads caused by frequent preemption and migration.

6.3.1. Proportionate Fairness Algorithms. The *Proportionate Fair* (Pfair) algorithm was introduced by Baruah et al. [1996]. Pfair is a schedule generation algorithm which is applicable to periodic tasksets with implicit deadlines. Pfair is based on the idea of *fluid* scheduling, where each task makes progress proportionate to its utilization (or weight in Pfair terminology). Pfair scheduling divides the timeline into equal length quanta or slots. At each time quanta t, the schedule allocates tasks to processors, such that the accumulated processor time allocated to each task τ_i will be either $\lceil tu_i \rceil$ or $\lfloor tu_i \rfloor$. Baruah et al. [1996] showed that the Pfair algorithm is optimal for periodic tasksets with implicit deadlines, and so has a utilization bound of

$$U_{PFAIR} = m. \tag{72}$$

In practice, however, the Pfair algorithm incurs very high overheads by making scheduling decisions at each time quanta. Further, all processors need to synchronize on the boundary between quanta when scheduling decisions are taken.

A number of variants on the Pfair approach have been introduced, including ERFair [Anderson and Srinivasan 2000], PD [Baruah et al. 1995], and PD^2 [Anderson and Srinivasan 2001]. The Pfair algorithm ensures that the $lag(\tau_i, t)$, given by the amount of execution time that should ideally have been allocated to task τ_i by time t (i.e., tu_i) less the processing time actually allocated, lies strictly between -1 and $+1$ (i.e., $-1 < lag < 1$). ERFair lifts the restriction that this lag must be greater than -1, thus allowing quanta of a job to execute before their PFair scheduling windows provided that the previous quanta of the same job has completed execution. This makes ERFair a work-conserving algorithm, whereas Pfair is not. PD [Baruah et al. 1995], and PD^2 [Anderson and Srinivasan 2001] improved on the efficiency of Pfair by separating tasks into groups of *heavy* ($u_i > 0.5$) and *light* tasks.

Anderson and Srinivasan [2000] extended the PFair approach to sporadic tasksets, showing that the EPDF (*earliest pseudodeadline first*) algorithm, a variant of PD, is optimal for sporadic tasksets with implicit deadlines executing on two processors, but is not optimal for more than two processors.

Zhu et al. [2003] introduced the *Boundary Fair* (BF) algorithm. Zhu et al. [2003] recognized that implicit-deadline tasks can only miss deadlines at times which are period boundaries. The BF algorithm is similar to Pfair; however, it only makes scheduling decisions at period boundaries. At any such time t^b, the difference between $t^b u_i$ and the accumulated processor time allocated to each task τ_i is again less than one time unit. In this sense BF is fair, but less fair than Pfair, as BF ensures only that proportionate progress is made on all tasks at period boundaries, but not at other times. Zhu et al. [2003] proved that BF is also an optimal algorithm for periodic tasksets with implicit deadlines, and showed via an empirical evaluation that the number of scheduling points is typically 25–50% of the number required for PD.

Holman and Anderson [2005] implemented Pfair scheduling on a symmetric multiprocessor. They found that the synchronized rescheduling of all processors every time quanta caused significant bus contention due to data being reloaded into cache. To address this problem, Holman and Anderson [2005] developed a variant of Pfair which staggers the time quanta on each processor. This reduces bus contention, at the cost of a reduction in schedulability. A task requiring a quanta every b slots, under Pfair, will require a quanta every $b - 1$ slots with the staggered approach.

6.3.2. LLREF. Cho et al. [2006] introduced the LLREF algorithm, which is also optimal for periodic tasksets with implicit deadlines. LLREF is based on the fluid scheduling model, using a T-L plane abstraction. LLREF divides the timeline into sections separated by normal scheduling events, that is, task releases, and coincident deadlines. At the start of each section, m tasks are selected to execute on the basis of *largest local remaining execution time first* (LLREF). The local remaining execution time for task τ_i at the start of section k is the amount of execution time that the task would be allocated during that section in a fluid schedule, that is, $t_f^k u_i$, where t_f^k is the length of the section. The local remaining execution time decrements as a task executes during the section. LLREF gives rise to additional scheduling events either when a running task completes its local execution time, or a nonrunning task reaches a state where it has no local laxity. At these additional scheduling points, the m tasks with the largest local remaining execution time are again selected to execute. Cho et al. [2006] showed that LLREF introduces at most an additional n scheduling events per section, giving a total of at most $n + 1$ scheduling events per task release.

Funaoka et al. [2008] extended the LLREF approach, apportioning processing time that would otherwise be unused among the tasks, and reapportioning processing time when a task completes earlier than expected, thus creating a work-conserving algorithm. Funaoka et al. [2008] showed that for taskset utilizations below 100% this approach results in significantly fewer preemptions than LLREF.

Funk and Nadadur [2009] extended the LLREF approach, forming the LRE-TL algorithm. The key observation of Funk and Nadadur [2009] was that, within each section, there is no need to select tasks for execution based on largest local remaining execution time. In fact, any task with remaining local execution time will do. This observation greatly reduces the number of migrations per section, compared to LLREF. Funk and Nadadur [2009] also showed how the LRE-TL algorithm could be applied to sporadic tasksets and proved that it is optimal (utilization bound of 100%) for sporadic tasksets with implicit deadlines.

6.3.3. EDZL. Lee [1994] introduced the *Earliest Deadline until Zero Laxity* (EDZL) algorithm, and showed that it dominates global EDF scheduling. Indeed, EDZL results in the same schedule as EDF until a situation is reached when a task will miss its deadline unless it executes for all of the remaining time up to its deadline (zero laxity). EDZL gives such a task the highest priority. Lee [1994] also showed that EDZL is

suboptimal for two processors (see also Cho et al. [2002] and Park et al. [2005]). Here, *suboptimal* is used to mean that EDZL can "schedule any feasible set of ready tasks." This weak form of optimality is appropriate for online scheduling algorithms, which cannot take account of future arrival times. Piao et al. [2006] showed that EDZL is completion time predictable in the sense defined by Ha and Liu [1994] (see Section 4.4). A simpler proof of predictability was given by Cirinei and Baker [2007], who also developed a sufficient schedulability test for EDZL based on the fundamental strategy of Baker [2003]: *a sporadic task system is schedulable by EDZL on m identical processors unless the following condition holds for at least m + 1 tasks and it holds strictly (>) for at least one of them:*

$$\sum_{\forall i \neq k} \beta_k^i \geq m(1 - \lambda_k), \tag{73}$$

where, $\lambda_k = C_i/\Delta_k$ and $\Delta_k = \min(Di, Ti)$ and

$$\beta_k^i = \frac{n_i C_i + \min(C_i, \max(0, \Delta_k - n_i T_i))}{\Delta_k}$$

Baker et al. [2008] refined the above sufficient test, and also gave an iterative sufficient test for EDZL based on the approach taken by Bertogna et al. [2008] for work-conserving algorithms and EDF. The test given by Equation (73) suffers from an overestimation of the amount of carry-in interference, particularly for tasksets with cardinality $n \gg m$. The iterative test of Baker et al. [2008] reduces this problem by calculating a lower bound on the slack for each task, and then using this value to limit the amount of carry-in interference and hence calculate a new value for the task slack. The empirical evaluation by Baker et al. [2008] shows that this iterative test for EDZL outperforms previous tests given in Cirinei and Baker [2007] and (as expected) similar tests for global EDF.

Kato and Yamasaki [2008c], introduced *Earliest Deadline until Critical Laxity*, effectively a variant of EDZL, which increases job priority on the basis of critical laxity at the release or completion time of a job. This has the effect of reducing the maximum number of context switches to two per job, the same as EDF, at the expense of slightly inferior schedulability, when compared to EDZL. Kato and Yamasaki [2008c] also corrected a minor flaw in the schedulability test for EDZL given by Cirinei and Baker [2007].

Chao et al. [2008] showed that the utilization bound for EDZL, assuming tasksets with implicit deadlines and large m, is

$$U_{EDZL} \leq m(1 - 1/e) \approx 0.63m, \tag{74}$$

where e is Euler's number 2.718.

7. HYBRID APPROACHES

Depending on the hardware architecture, the overheads incurred by global scheduling can potentially be very high. The fact that jobs can migrate from one processor to another can result in additional communication loads and cache misses, leading to increased worst-case execution times that would not occur in the fully partitioned/nonmigration case. However, fully partitioned approaches suffer from the drawback that the available processing capacity can become fragmented, such that, although in total a large amount of capacity is unused, no single processor has sufficient capacity remaining to schedule further tasks. Indeed, the maximum utilization bound is just 50% of the total processing capacity. In this section we outline recent research into hybrid approaches which combine elements of both partitioned and global scheduling.

7.1. Semipartitioned Approaches

One approach aimed at addressing the fragmentation of spare capacity in partitioned systems is to split a small number of tasks between processors.

Andersson and Tovar [2006] introduced *EKG*, an approach to scheduling periodic tasksets with implicit deadlines, based on partitioned scheduling, but splitting some tasks into two components that execute at different times on different processors. Andersson and Tovar [2006] showed that the utilization bound for EKG depends on the parameter k, used to control division of tasks into groups of heavy and light tasks. The utilization bound for EKG is given by

$$U_{EKG} = \begin{cases} k/(k+1), & k < m, \\ 1, & k. \end{cases} \tag{75}$$

Hence the utilization bound is 100% for $k = m$. Further, the average number of preemptions per job over the hyperperiod is bounded by $2k$. Thus, as suggested by Andersson and Tovar [2006], choosing a value of $k = 2$ gives a utilization bound of 66% and at most an average of four preemptions per job.

Andersson and Bletsas [2008] developed the idea of job splitting to cater for sporadic tasksets with implicit deadlines. In this case, each processor p executes at most two split tasks, one also executed by processor $p - 1$ and one also executed by processor $p + 1$. Andersson et al. [2008] latter extended this approach to tasksets with arbitrary deadlines. They showed that first fit and next fit are not good allocation strategies when task splitting is employed. Instead, they order tasks by decreasing relative deadline and try to fit all tasks on the first processor before then choosing the remaining task with the shortest relative deadline to be split. At runtime, the split tasks are scheduled at the start and end of fixed-duration time slots. The disadvantage of this approach is that the capacity required for the split tasks is inflated if these slots are long, while the number of preemptions is increased if the time slots are short. In the implicit deadline case, Andersson and Bletsas [2008] showed that this approach has a utilization bound of

$$U = 4(\sqrt{\delta(\delta+1)} - \delta) - 1, \tag{76}$$

where δ effectively defines the slot length ($T_{\min}/\delta$). This utilization bound equates to approximately 88% for $\delta = 4$. Further, the number of additional preemptions, per processor, in an interval of length t is given by

$$3\delta \lceil t/T_{\min} \rceil + 2. \tag{77}$$

Bletsas and Andersson [2009] developed an alternative approach based on the concept of *notional processors*. With this method, tasks are first allocated to physical processors (heavy tasks first) until a task is encountered that cannot be assigned. Then the workload assigned to each processor is restricted to periodic reserves and the spare time slots between these reserves organized to form notional processors. (A notional processor is formed from time slots on a number of physical processors which taken together provide continuous execution capacity.) Bletsas and Andersson [2009] showed that this method has a utilization bound of at least 66.6% for tasksets with implicit deadlines, and that the total number of additional preemptions, beyond those caused by task arrivals, is given by

$$\left(2m + \left\lceil \frac{m}{3} \right\rceil\right)\left\lceil \frac{t}{S} \right\rceil, \tag{78}$$

where t is the length of the time interval, and S is the minimum period of any task on the processor considered.

Kato and Yamasaki [2007] introduced the *Ehd2-SIP* algorithm. Ehd2-SIP is predominantly a partitioning algorithm, with each processor scheduled according to an algorithm based on EDF; however, Ehd2-SIP splits at most $m - 1$ tasks into two portions to be executed on two separate processors. EhD2-SIP has a utilization bound of 50%.

Kato and Yamasaki [2008a] presented a further semi-partitioning algorithm called *EDDP*, also based on EDF. EDDP again splits at most $m - 1$ tasks across two processors. The two portions of each split task are prevented from executing simultaneously by EDDP, which instead defers execution of the portion of the task on the lower numbered processor, while the portion on the higher numbered processor executes. During the partitioning phase, EDDP places each heavy task with utilization greater than 65% on its own processor. The light tasks are then allocated to the remaining processors, with at most $m - 1$ tasks split into two portions. Kato and Yamasaki [2008a] showed that EDDP has a utilization bound of 65% for periodic tasksets with implicit deadlines, and performs well in terms of the typical number of context switches required, which is less than that of EDF due to the placement strategy for heavy tasks. Subsequently, Kato and Yamasaki [2008b] also extended this approach to fixed-task priority scheduling, showing that the RMDP algorithm has a utilization bound of 50%.

Kato and Yamasaki [2009] developed a semipartitioning algorithm called *DM-PM* (Deadline-Monotonic with Priority Migration), applicable to sporadic tasksets, and using fixed-priority scheduling. DM-PM strictly dominates fully partitioned fixed-task priority approaches, as tasks are only permitted to migrate if they won't fit on any single processor. Tasks chosen for migration are assigned the highest priority, with portions of their execution time assigned to processors, effectively filling up the available capacity of each processor in turn. At runtime, the execution of a migrating task is staggered across a number of processors, with execution beginning on the next processor once the portion assigned to the previous processor completes. Thus no job of a migrating task returns to a processor it has previously executed on. Kato and Yamasaki [2009] showed that DM-PM has a utilization bound of 50% for tasksets with implicit deadlines. Subsequently, Kato et al. [2009] extended the same basic approach to EDF scheduling, forming the *EDF-WM* algorithm (EDF with window-constrained migration).

Lakshmanan et al. [2009] developed a semipartitioning method based on fixed-priority scheduling of sporadic tasksets with implicit or constrained deadlines. This method, called *PDMS_HPTS*, splits only a single task on each processor: the task with the highest priority. Note that a split task may be chosen again for splitting if it has the highest priority on another processor. PDMS_HPTS takes advantage of the fact that, under fixed-priority preemptive scheduling, the response time of the highest-priority task on a processor is the same as its worst-case execution time, leaving the maximum amount of the original task deadline available for the part of the task split on to another processor. Lakshmanan et al. [2009] showed that for any task allocation PDMS_HPTS has a utilization bound of at least 60% for tasksets with implicit deadlines; however, if tasks are allocated to processors in order of decreasing density (PDMS_HPTS_DS), then this bound increases to 65%. Further, PDMS_HPTS_DS has a utilization bound of 69.3% if the maximum utilization of any individual task is no greater than 0.41. Notably, this is the same as the Liu and Layland [1973] bound for single-processor systems, without the restriction to individual task utilization.

7.2. Clustering

Clustering can be thought of as a form of partitioning with the clusters effectively forming a smaller number of faster processors to which tasks are allocated. Thus capacity fragmentation is less of an issue than with partitioned approaches, while the small number of processors in each cluster reduces global queue length and has

the potential to reduce migration overheads, depending on the particular hardware architecture. For example, processors in a cluster may share the same cache, reducing the penalty in terms of increased worst-case execution time, of allowing tasks to migrate from one processor to another.

Shin et al. [2008] derived schedulability analysis for multiprocessor systems, where tasks are allocated to clusters of processors and scheduled according to global EDF on processors within their cluster. Clusters are represented by a multiprocessor periodic resource (MPR) abstraction and may be either physical, with a static mapping to processors, or virtual, with a dynamic mapping to processors. Shin et al. [2008] developed a hierarchical scheduling model and analysis appropriate to tasks executing within MPRs which are then scheduled on the multiple processors. The algorithm proposed was shown to be optimal for tasksets with implicit deadlines; however, the maximum number of preemptions which can take place is $m - 1$ in an interval equal to the GCD (greatest common divisor) of the task periods. We note that, in practice, this number of context switches can be prohibitive.

Leontyev and Anderson [2008] developed a container-based hierarchical scheduling scheme for multiprocessor systems executing both hard and soft real-time tasks. Here, each container is allocated a specific bandwidth, which is provided via minimum parallelism, using the maximum number of fully utilized processors and at most one processor which is partially utilized. This partial bandwidth is provided by a periodic server. Leontyev and Anderson [2008] showed how the tardiness of soft real-time tasks can be bounded in this model, without any loss of utilization. Utilization loss does occur when hard-real-time tasks are included; however, this loss was shown to be small provided that the utilization of hard real-time tasks is a small fraction of the total.

8. RESOURCE SHARING

The previous sections described partitioned, global, and hybrid scheduling algorithms and analyses for simple periodic and sporadic task models where the execution of one task is independent of the others. In this section, we discuss research lifting this assumption of independence and therefore allowing tasks to share resources that have to be accessed in mutual exclusion.

We note that with parallel access to shared resources there are a number of alternatives to the classic lock-based schemes. *Lock-free* algorithms [Anderson et al. 1997], which are similar to those used in optimistic concurrency control in database systems, allow immediate access to the resource, but latter check to see if there was a conflict over this access. If there was, then computations are abandoned and the resource is reaccessed. These algorithms are therefore lock-free, but can involve looping while the task is waiting to gain conflict-free access. Another class of algorithms is termed *wait-free*. Wait-free algorithms involve neither locks nor loops but multiple copies of the data are required. An example of a wait-free scheme is the four-slot mechanism [Simpson 1990], which preserves independence of execution of a single reader and a single writer over a shared data resource. This mechanism does, however, require memory space for four copies of the data.

In uniprocessor systems, the Stack Resource Policy (SRP) [Baker 1991] and Priority Ceiling Protocol (PCP) [Sha et al. 1990] are widely accepted as the most appropriate mechanisms to use to provide access to mutually exclusive shared resources. Initial research into suitable policies for multiprocessor real-time systems has built on these uniprocessor lock-based protocols.

8.1. Partitioned Scheduling

Rajkumar et al. [1988] introduced a multiprocessor variant of the Priority Ceiling Protocol called *MPCP*, which is applicable to partitioned systems using fixed priorities.

Under MPCP, the priority ceilings of global shared resources are set to levels that are strictly higher than that of any task in the system. At runtime when a task attempts to access a locked global resource, it is suspended, and waits in a prioritized queue associated with the resource. This allows lower-priority local tasks to continue executing. When the resource is unlocked, then the task at the head of the queue waiting on it is resumed and executes at the ceiling priority of the resource.

Allowing low-priority tasks to execute while a higher-priority task on the same processor is blocked on a global resource has the important effect of permitting further priority inversion. The low-priority task can attempt to access another locked global resource with a higher ceiling and can therefore subsequently execute ahead of the high-priority task even when the original resource is unlocked. MPCP has the restrictions that nested access to globally shared resources is not permitted, and that nesting of local and global critical sections is not permitted. MPCP provides a bounded blocking time, with a sufficient schedulability test based on the utilization bound of Liu and Layland [1973]. The blocking factor is made up of five different components, which are summarized by Gai et al. [2003].

Chen et al. [1994] described a further variant of PCP called *MDPCP*, and provided a simple sufficient test for partitioned EDF using this protocol. This test is based on computing blocking factors due to four different types of blocking.

Gai et al. [2001] introduced the *MSRP* protocol based on SRP [Baker 1991]. MSRP is again applicable to partitioned systems, using either fixed priorities or EDF. A significant difference between MSRP and MPCP is that, when a task is blocked on a global resource under MSRP, it busy waits and is not preemptable. This behavior is referred to as a *spin-lock*. A FIFO queue is again used to grant access to tasks waiting on a global resource when it is unlocked. MSRP provides both a bounded blocking time and bounded increases in task execution times due to the spin-locks. MSRP can also be analyzed using a simple sufficient schedulability test. Under MSRP, task execution on each processor is perfectly nested and so the tasks can share a single stack.

In comparison with MPCP, MSRP removes two of the five contributions to the blocking factor; however, the spin-locks consume processing time, which could otherwise be used by other tasks. Further, MSRP has the advantage that it is significantly simpler to implement than MPCP. A study performed by Gai et al. [2003] showed that MSRP typically outperforms MPCP when global critical sections are short and access to local resources dominates access to global resources.

8.2. Global Scheduling

Devi et al. [2006], considered the problem of accessing mutually exclusive shared resources under global EDF scheduling. They suggested two simple approaches for short nonnested access to shared data structures: *spin-based queue locks* [Mellor-Crummey and Scott 1991] and lock-free synchronization. With spin-based queue locks, tasks waiting for access to a resource busy-wait on a "spin variable" which is exclusive to that task. When a task exits the resource, it updates the spin variable of the next task in the queue. In the first approach suggested by Devi et al. [2006], the spin queue grants access to resources in FIFO order, and further access to each resource is nonpreemptable; hence the longest time for which a task can be blocked waiting to access a global shared resource, with access time e, on an m processor system is $(\min(m, c) - 1)e$, where c is the number of tasks that access the resource. With lock-free synchronization, operations on shared resources (data structures) are implemented as "retry-loops"; thus operations are opportunistically attempted and, if there is contention, then they are retried until they are successful. Devi et al. [2006] showed how simple schedulability tests for global EDF can be modified to take account of the effects of spin-based queue locking and lock-free synchronization using retries. The performance evaluation

reported by Devi et al. [2006] suggests that the total overheads of spin-based queue locks are significantly less than those of lock-free synchronization.

Block et al. [2007] introduced the *Flexible Multiprocessor Locking Protocol* (FMLP). FMLP operates using a variant of global EDF (or other algorithms), which ensures that a job can only be blocked by another nonpreemptable job when it is released or resumed. FMLP divides resources into two types with *long* and *short* access times. Jobs waiting to access a short resource do so by becoming nonpreemptable and busy-waiting. Jobs waiting to access long resources do so by blocking on a semaphore queue, in which case the job currently accessing the resource inherits the priority of the highest-priority job in the queue. FMLP uses a simple method of avoiding deadlock by grouping resources that can be nested and ensuring that only a single job can access the resources in a group at any given time. FMLP has the advantage that it can handle nested resource access without the requirement for tasks accessing nested resources to be allocated the same processor, as is the case with MSRP. Further, FMLP optimizes the simple case of nonnested access to short resources. Block et al. [2007] showed via some simple experiments that FMLP has better performance than MSRP. This advantage is at least partly due to the fact that FMLP removes the restriction on task allocation required by MSRP.

Brandenburg et al. [2008b] examined the relative performance of blocking and nonblocking approaches to accessing shared resources. The blocking approaches used FMLP and considered both spinning, that is, busy-waiting, and suspending. The nonblocking approaches considered were *lock-free* and *wait-free*. Brandenburg et al. concluded that nonblocking approaches are preferable for small and simple resource objects and wait-free or spin-based algorithms are generally preferable for more complex resource objects with longer access times. Suspension-based algorithms are almost never better than spin-based variants.

9. EMPIRICAL INVESTIGATIONS

In this section, we review recent empirical investigations into the performance of multiprocessor scheduling algorithms and their associated schedulability tests.

9.1. Schedulability Test Performance

As well as producing theoretical results including approximation ratios, utilization bounds, and speedup factors, many researchers have also used empirical methods to investigate the relative performance of different real-time scheduling algorithms and their analyses. The most commonly used metric is the number of randomly generated tasksets that are deemed schedulable. This empirical metric is an important one in real-time systems research. For techniques to be transferred into industrial practice, it is essential that they are both simple and efficient, as well as highly effective for the majority of realistic cases. While utilization/density-based tests and speedup factors are useful performance indicators, they focus heavily on specific pathological tasksets. By comparison, more general schedulability tests that take into account the parameters of individual tasks have the potential to provide superior performance in the vast majority of cases, something that is highlighted by empirical studies.

In empirical studies, parameters such as the number of tasks, the number of processors, taskset utilization, the range of task periods, the distribution of task deadlines, and the distribution of individual task utilizations can be varied to examine the performance of the algorithms and their schedulability tests over a range of different credible scenarios.

Baker [2006b] made an empirical comparison between the best global EDF, and partitioned EDF scheduling algorithms available in 2006. The empirical performance measure used was the number of randomly generated tasksets that were schedulable

according to each algorithm. The conclusion of this study was that, although the two approaches are incomparable, the partitioned approach appeared to outperform the global approach on this metric by a significant margin.

Considering global scheduling algorithms, in the simulation chapter of his thesis, Bertogna [2007] showed that the iterative response time test for global FP scheduling (Equation (63)) outperformed all other tests for global FP and global EDF scheduling and also similar tests for EDZL (see Section 6.3.3) known at the time. Since real-time system designers are interested in provable schedulability, Bertogna [2007] argued that global FP scheduling can reasonably be regarded as one of the best global scheduling techniques to use as it is simple to implement and is supported by a demonstrably effective schedulability test.

Bertogna [2009] investigated the performance of the following schedulability tests for global EDF:

—GFB, density-based test [Goossens et al. 2003];
—BAK [Baker 2005];
—BAR [Baruah 2007];
—LOAD, processor load-based test [Baruah and Baker 2009];
—BCL, [Bertogna et al. 2005b];
—RTA, response time analysis test [Bertogna and Cirinei 2007];
—FF-DBF, speedup optimal test [Baruah et al. 2009].

Bertogna [2009] showed that, of these tests, the RTA test [Bertogna and Cirinei 2007] was the most effective in terms of the number of randomly generated tasksets deemed to be schedulable, although the RTA test can only be shown to strictly dominate the BCL test, and is incomparable with all of the other tests listed above. Bertogna [2009] also sequentially applied the RTA test, the BAR test and the FF-DBF test to form a composite test (COMP). This test utilizes intermediate information from the RTA test, when it fails to show schedulability, to improve the performance of the BAR test. The COMP test was shown to improve upon the performance of the RTA test.

Davis and Burns [2009] showed that, in global FP scheduling, the number of randomly generated tasksets deemed schedulable using the schedulability tests of Bertogna et al. [2008] is significantly increased by using Audsley's [1991, 2001] optimal priority assignment policy rather than deadline monotonic priority assignment. The latter policy, although optimal for uniprocessor systems, was shown to perform poorly in the multiprocessor case. Davis and Burns [2009] also showed that DkC is a highly effective priority assignment policy for global FP schedulability tests that are not compatible with the optimal priority assignment algorithm. Again, performance was significantly better than with deadline monotonic priority assignment.

Davis and Burns [2009] also showed how the UUnifast method of taskset generation [Bini and Buttazzo 2005], which is the de facto standard for investigation of schedulability test performance in uniprocessor systems, could be extended to the multiprocessor case. The resulting method, called *UUnifast-Discard*, generates tasksets with specific parameter settings, thus facilitating an empirical study of schedulability test effectiveness without the problem of confounding variables. We note that the method of taskset generation used in Bertogna and Cirinei [2007] and Bertogna [2009], while valid in comparing the performance of different schedulability tests, suffers from a problem of confounding variables: as taskset utilization is increased, the average cardinality of the tasksets generated increases, effectively linking these two variables and so obscuring their individual influences on schedulability test effectiveness.

While empirical studies of schedulability test performance, such as those described above, provide important information about the theoretical effectiveness of different algorithms and their schedulability tests, they leave an important question unanswered.

How does this theoretical performance translate in practice, when the overheads involved in scheduling decisions, context switches, and migration are considered?

9.2. Measurements

Brandenburg et al. [2008a] measured the performance of various scheduling algorithms and their overheads on a LITMUS testbed using a Sun UltraSPARC Niagra multicore platform with 32 logical processors (actually four hardware threads running on each of eight CPUs). Brandenburg et al. [2008a] examined *partitioned*, *clustered*, and *global* approaches using EDF and Pfair algorithms. They found that the overheads of pure Pfair meant it had very poor performance, while staggered Pfair performed much better in practice. Global EDF scheduling performed poorly due to the overheads involved in manipulating a lengthy global queue, accessible to all processors. Partitioned EDF was shown to work best for hard real-time tasksets, except when the tasks had high individual utilizations, when staggered Pfair was best. For soft real-time tasksets, partitioned EDF was again effective unless the tasks had high individual utilizations >0.5. Clustered EDF was also highly effective for soft-real time tasksets. The key point that can be drawn from this work is that overheads are a significant issue for multiprocessor real-time scheduling.

10. SUMMARY, OPEN ISSUES, AND DIRECTION FOR FUTURE RESEARCH

Although research into multiprocessor real-time scheduling and schedulability analysis has advanced markedly since the seminal paper of Dhall and Liu [1978], there are still significant and fundamental research challenges that remain. Global, clustered, and semipartitioned approaches to multiprocessor scheduling offer potential solutions for future complex high-performance real-times systems; however, few results can be identified in these areas that are ready to be transferred into industrial practice.

10.1. Open Issues

The following are a selection of key open issues with existing research into multiprocessor real-time scheduling identified by this survey.

Limits on processor utilization. Fixed-job-priority and partitioning algorithms are in the worst case capable of utilizing only 50% of the available processing resource. While global dynamic algorithms can in some cases utilize up to 100%, their overheads are typically prohibitive. Further research is needed into minimally dynamic algorithms, and novel approaches to partitioning task execution that can increase guaranteed processing capability without introducing significant overheads. Recent progress in this area is summarized in Section 7.

Ineffective schedulability tests. For the sporadic task model, empirical studies have shown that a large gap exists between the best sufficient schedulability tests currently available for global fixed-job-priority and fixed-task priority scheduling and what may be possible as indicated by feasibility/infeasibility tests. Closing this gap is a key area for future research. Fundamental to this problem is the fact that "no finite collection of worst-case job arrival sequences has been identified for the global scheduling of sporadic task systems" [Baruah 2007, page 121].

Consideration of overheads. Advanced hardware features, such as cache architectures, have a large impact on the cost of migration (at the task and job levels). Recent experimental implementations [Brandenburg et al. 2008a] on multiprocessor platforms have shown that the overheads of migration, context switching, and run-queue manipulation are a key issue for multiprocessor scheduling. Research into scheduling algorithms and analysis needs to take appropriate account of such overheads. Further

research is needed into algorithms that permit only task-level migration, or only permit migration within a limited cluster of processors.

Limited task models for multiprocessor systems. The vast majority of existing research into hard real-time scheduling on multiprocessors has addressed simple periodic or sporadic task models originally developed with uniprocessor systems in mind. More general task models are needed that can express both the benefits and overheads of executing parts of the same task in parallel. Initial work in this area by Collette et al. [2007, 2008] has considered the work-limited job parallelism of each task defined by the rate at which it can execute on 1 to m processors. Another relevant model is the task model of Edmonds and Pruhs [2009], which considers each task as being made up of a number of phases each of which has an amount of computation that must be completed in that phase and a speedup function indicating how the rate at which that computation is executed increases with the degree of parallelism (number of processors executing the phase). Work in the area of online scheduling methods covering scalable tasks [Lee et al. 2003], and the application of divisible load theory [Bharadwaj et al. 1996; Robertazzi 2003; Veeravalli et al. 2003; Lin et al. 2007] are also of interest in this respect.

As well as more expressive task models, more general models of processing supply would provide a means of abstracting away from specific hardware platforms to a virtual platform model, thus enabling composition. Initial work in this area by Bini et al. [2009a, 2009b] has modeled the parallel supply of a virtual platform as a set of m supply functions indicating how the minimum supply of processing capacity on 1 to m processors varies as a function of time.

Limited policies for access to shared resources. Unlike uniprocessor systems, where the Stack Resource Policy [Baker 1991] is widely accepted as the most effective protocol to use to control mutually exclusive accesses to shared resources, there is no such consensus for multiprocessor scheduling. Research in this area has indicated that spin-based approaches appear to be preferable to suspension-based methods; however, nonblocking approaches also perform well for simple resource accesses. It therefore seems unlikely that there will be a single best solution here. Different forms of resource sharing and different architectures are likely to require different forms of support.

10.2. Related Areas of Research

This survey covers research into hard real-time scheduling for homogeneous multiprocessor systems. There are a number of related areas that, while outside the scope of the survey, are likely to be of interest to researchers and practitioners developing multiprocessor real-time systems. These include the following.

—Worst-case execution time (WCET) analysis;
—Network/bus scheduling;
—Memory architectures;
—Scheduling of uniform and heterogeneous processors;
—Operating systems;
—Power consumption and dissipation;
—Scheduling tasks with soft real-time constraints, and
—Non-real-time issues such as load balancing.

In particular, WCET analysis of multicore systems presents a significant challenge. Here, even with the simplest partitioned approaches to task allocation, tasks executing on one processor can indirectly affect the WCET of tasks on other processors by contending for shared hardware resources such as cache and network-on-chip [Quinones et al. 2009]. A recent survey of WCET techniques [Wilhelm et al. 2008] concluded that no static analysis currently exists for multicore processors which have such complex

hardware interactions. Research in this area has either aimed to develop a fully statically analyzable multicore CPU, or to move from providing deterministic WCET estimates to providing probabilistic ones with a high degree of confidence [PROARTIS 2009].

As well as issues fundamental to multiprocessor scheduling set out in the previous section, the confluence of WCET analysis and scheduling theory for multicore systems represents a key area for future research. For example, the effects on task execution time caused by contention for hardware resources, due to parallel execution of other tasks, can be modeled as a scheduling effect or as an increase in WCET. At this level, these two areas of research are closely interwoven.

11. CONCLUSIONS

Currently, progress in developing multiprocessor systems is a long way ahead of research efforts to determine the best mechanisms, policies, and analysis to use in these systems. At best, this can result in systems that are heavily overspecified and expensive; at worst, it can lead to intermittent and unexpected timing faults that compromise system reliability. Functionality, unit cost, time-to-market, and a reputation based on product reliability are key factors for companies developing real-time embedded systems. All of these factors can be compromised by building systems using approaches that lack the necessary theoretical underpinnings. Ultimately, multiprocessors will be used in high-integrity real-time systems, and consequently, timing failures could affect safety.

Future advances along the research directions indicated in this survey should help resolve the key open issues identified. These advances hold the promise of providing the effective and efficient mechanisms, policies, and analyses required for a sound engineering-based approach to the development of complex commercial multiprocessor real-time systems.

ACKNOWLEDGMENTS

The authors would like to thank Sanjoy Baruah and Dirk Müller for their comments and suggestions on an earlier draft.

REFERENCES

ANDERSON, J., RAMAMURTHY, S., AND JEFFAY, K. 1997. Real-time computing with lock-free shared objects. *ACM Trans. Comp. Syst. 15*, 2, 134–165.

ANDERSON, J. AND SRINIVASAN, A. 2000. Early-release fair scheduling. In *Proceedings of the Euromicro Conference on Real-Time Systems*.

ANDERSON, J. AND SRINIVASAN, A. 2000b. Pfair scheduling: Beyond periodic task systems. In *Proceedings of the 7th International Workshop on Real-Time Computing Systems and Applications*.

ANDERSON, J. AND SRINIVASAN, A. 2001a. Mixed Pfair/Erfair scheduling of asynchronous periodic tasks. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*.

ANDERSSON, B. AND JONSSON, J. 2000a. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition, In *Proceedings of the International Workshop on Real-Time Computing Systems and Applications*.

ANDERSSON, B. AND JONSSON, J. 2000b. Some insights on fixed-priority pre-emptive non-partitioned multiprocessor scheduling. In *Proceedings of the Real-Time Systems Symposium—Work-in-Progress Session*.

ANDERSSON, B., BARUAH, S., AND JONSSON, J. 2001. Static-priority scheduling on multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*. 193–202.

ANDERSSON, B. 2003. Static-priority scheduling on multiprocessors. Ph.D. dissertation. Chalmers University of Technology, Gothenburg, Sweden.

ANDERSSON, B. AND JONSSON, J. 2003. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. *In Proceedings of the 15th Euromicro Conference on Real-Time Systems*.

ANDERSSON, B. AND TOVAR, E. 2006. Multiprocessor scheduling with few preemptions. In *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*.

ANDERSSON, B., BLETSAS, K., AND BARUAH, S. K. 2008. Scheduling arbitrary deadline sporadic task systems on multiprocessors. In *Proceedings of the Real-Time Systems Symposium*. 384–393.

ANDERSSON, B. AND BLETSAS, K. 2008. Sporadic multiprocessor scheduling with few preemptions. In *Proceedings of the Euromicro Conference on Real-Time Systems*. 243–252.

ANDERSSON, B. 2008. Global static-priority preemptive multiprocessor scheduling with utilization bound 38%. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*.

AUDSLEY, N. C. 1991. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Tech. rep. YCS 164. Dept. Computer Science, University of York, York, U.K.

AUDSLEY, N. C., BURNS, A., DAVIS, R. I., TINDELL, K. W., AND WELLINGS, A. J. 1996. Fixed priority scheduling: an historical perspective. *Real-Time Syst. 8*, 3, 173–198.

AUDSLEY, N. C. 2001. On priority assignment in fixed priority scheduling. *Inform. Process. Lett. 79*, 1, 39–44.

BAKER, T. P. 1991. Stack-based scheduling of real-time processes. *Real-Time Syst. J. 3*, 1, 67–100.

BAKER, T. P. 2003. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*. 120–129.

BAKER, T. P. 2005. An analysis of EDF scheduling on a multiprocessor. *IEEE Trans. Parall. Distrib. Syst. 15*, 8, 760–768.

BAKER, T. P. 2006a. An analysis of fixed-priority scheduling on a multiprocessor. *Real Time Syst. 32*, 1-2, 49–71.

BAKER, T. P. 2006b. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Proceedings of the International Conference on Real-Time and Network Systems*. 119–130.

BAKER, T. P. AND CIRINEI, M. 2006. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In *Proceedings of the Work-In-Progress (WIP) Session of the 27th IEEE Real-Time Systems Symposium*.

BAKER, T. P. AND BARUAH, S. K. 2007a. Schedulability analysis of multiprocessor sporadic task systems. In *Handbook of Real-Time and Embedded Systems*, I. Lee, Joseph Y.-T. Leung, and S. H. Son, Eds., Chapman Hall/CRC Press.

BAKER, T. P. AND BARUAH, S. K. 2007b. Schedulability analysis of multiprocessor sporadic task systems. Tech. rep. TR-060601. Florida State University, Tallahassee, FL.

BAKER, T. P., CIRINEI, M., AND BERTOGNA, M. 2008. EDZL scheduling analysis. *Real-Time Syst. 40*, 3, 264–289.

BAKER, T. P. AND BARUAH, S. K. 2008. Schedulability analysis of global EDF. *Real-Time Syst. 38*, 223–235.

BAKER, T .P. AND BARUAH, S. K. 2009. Sustainable multiprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*. 141–150.

BARUAH, S. K., MOK, A. K., AND ROSIER, L. E. 1990a. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the Real-Time System Symposium*. 182–190.

BARUAH, S. K., ROSIER, L. E., AND HOWELL, R. R. 1990b. Algorithms and complexity: Concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Syst. 2*, 4, 301–324.

BARUAH, S. K., GEHRKE, J., AND PLAXTON, C. G. 1995. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the International Parallel Processing Symposium*.

BARUAH, S. K., COHEN, N., PLAXTON, G., AND VARVEL, D. 1996. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica 15*, 6, 600–625.

BARUAH, S. K. AND GOOSSENS, J. 2003. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Trans. Comp. 52*, 7, 966–970.

BARUAH, S. K. AND CARPENTER, J. 2003. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *Proceedings of the EuroMicro Conference on Real-Time Systems*. 195–202.

BARUAH, S. K. AND FISHER, N. 2005. Partitioned multiprocessor scheduling of sporadic task systems. *In Proceedings of the 26th Real-Time Systems Symposium*.

BARUAH, S. K. AND FISHER, N. 2006. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Trans. Comp. 55*, 7, 918–923.

BARUAH, S. K. AND BURNS, A. 2006. Sustainable scheduling analysis. In *Proceedings of the IEEE Real-Time Systems Symposium*. 159–168.

BARUAH, S. K. 2007. Techniques for multiprocessor global schedulability analysis. In *Proceedings of the Real-Time Systems Symposium*. 119–128.

BARUAH, S. K. AND FISHER, N. 2007. The partitioned dynamic-priority scheduling of sporadic task systems. *Real-Time Syst. 36*, 3, 199–226.

BARUAH, S. K. AND FISHER, N. 2008a. Non-migratory feasibility and migratory schedulability analysis of multiprocessor real-time systems. *Real-Time Syst. 39*, 1-3, 97–122.

BARUAH, S .K. AND FISHER, N. 2008b. Global fixed-priority scheduling of arbitrary-deadline sporadic task systems. In *Proceedings of the 9th International Conference on Distributed Computing and Networking*.

BARUAH, S. K. AND BAKER, T. P. 2008. Global EDF schedulability analysis of arbitrary sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*. 3–12.

BARUAH, S. K. AND BAKER, T. P. 2009. An analysis of global EDF schedulability for arbitrary sporadic task systems. *Real-Time Syst. 43*, 1, 3–24.

BARUAH, S. K., BONIFACI, V., MARCHETTI-SPACCAMELA, A., AND STILLER, S. 2009. Implementation of a speedup-optimal global EDF schedulability test. In *Proceedings of the EuroMicro Conference on Real-Time Systems*. 259–268.

BARUAH, S. K. 2007. Schedulability analysis of global deadline monotonic scheduling. Tech. rep., University of North Carolina, Chapel Hill, NC. `http://www.cs.unc.edu/~baruah/Pubs.shtml`.

BERTOGNA, M., CIRINEI, M., AND LIPARI, G. 2005a. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *Proceedings of the 9th International Conference on Principles of Distributed Systems*.

BERTOGNA, M., CIRINEI, M., AND LIPARI, G. 2005b. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*. 209–218.

BERTOGNA, M. 2007. Real-time scheduling analysis for multiprocessor platforms. Ph.D. dissertation. Scuola Superiore Sant'Anna, Pisa, Italy.

BERTOGNA, M. AND CIRINEI, M. 2007. Response time analysis for global scheduled symmetric multiprocessor platforms. In *Proceedings of the Real-Time Systems Symposium*. 149–158.

BERTOGNA, M., CIRINEI, M., AND LIPARI, G. 2008. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Trans. Paral. Distrib. Syst. 20*, 4, 553–566.

BERTOGNA, M. 2009. Evaluation of existing schedulability tests for global EDF. In *Proceedings of the 1st International Workshop on Real-Time Systems on Multicore Platforms: Theory and Practice*.

BHARADWAJ, V., GHOSE, D., MANI, V., AND ROBERTAZZI, T. G. 1996. *Scheduling Divisible Loads in Parallel and Distributed Systems.* IEEE Computer Society Press, Los Alamitos, CA.

BINI, E. AND BUTTAZZO, G. C. 2005. Measuring the performance of schedulability tests. *Real-Time Syst. 30*, 1–2, 129–154.

BINI, E., BUTTAZZO, G. C. AND BERTOGNA, M. 2009a. The multi supply function abstraction for multiprocessors. In *Proceedings of the Conference on Real-Time Computing Systems and Applications*.

BINI, E., BERTOGNA, M., AND BARUAH, S.K. 2009b. Virtual Multiprocessor Platforms: Specification and Use. In *Proceedings of the Real-Time Systems Symposium*.

BLETSAS, K. AND ANDERSSON, B. 2009. Notional processors: An approach for multiprocessor scheduling. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*. 3–12.

BLOCK, A., LEONTYEV, H., BRANDENBURG, B., AND ANDERSON, J. H. 2007. A flexible real-time locking protocol for multiprocessors. In *Proceedings of the Conference on Real-Time Computing Systems and Applications*. 47–56.

BRANDENBURG, B. B., CALANDRINO, J. M. AND ANDERSON, J. H. 2008a. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *Proceedings of the Real-Time Systems Symposium*. 157–169.

BRANDENBURG, B. B., CALANDRINO, J. M., BLOCK, A., LEONTYEV, H., AND ANDERSON, J. H. 2008b. Real-time synchronization on multiprocessors: to block or not to block, to suspend or spin. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*. 342–353.

BURCHARD, A., LIEBEHERR, J., OH, Y., AND SON, S. H. 1995. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Trans. Comp. 44*, 12, 1429–1442.

BURNS, A. AND WELLINGS, A. 2009. *Real-Time Systems and Programming Languages*, 4th ed. Addison Wesley Longmain, Reading, MA.

BONIFACI, V., MARCHETTI-SPACCAMELA, A., AND STILLER, S. 2008. A constant-approximate feasibility test for multiprocessor real-time scheduling. In *Proceedings of the European Symposium on Algorithms*. 210–221.

BUTTAZZO, G. 2005. *Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications,* 2nd ed., Springer, Berlin, Germany.

CARPENTER, J., FUNK, S., HOLMAN, P., SRINIVASAN, A., ANDERSON, J. H., AND BARUAH, S. K. 2004. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman & Hall, CRC Computer.

CHAO, Y.-H., LIN, S.-S., AND LIN, K.-J. 2008. Schedulability issues for EDZL scheduling on real-time multiprocessor systems. *Inform. Process. Lett. 107*, 5, 158–164.

CHEN, C-M., TRIPATHI, S.K., AND BLACKMORE, A. 1994. A resource synchronization protocol for multiprocessor real-time systems. In *Proceedings of the International Conference on Parallel Processing*. 159–162.

CHO S., LEE, S. K., HAN, A., AND LIN, K.-J. 2002. Efficient real-time scheduling algorithms for multiprocessor systems. *IEICE Trans. Comm. E85-B*, 12, 2859–2867.

CHO, H., RAVINDRAN, B., AND JENSEN, E. D. 2006. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the Real-Time Systems Symposium*. 1001–110.

CIRINEI, M. AND BAKER, T. P. 2007. EDZL scheduling analysis. In *Proceedings of the EuroMicro Conference on Real-Time Systems*. 9–18.

COLLETTE, S., CUCU, L., AND GOOSSENS, J. 2007. Algorithm and complexity for the global scheduling of sporadic tasks on multiprocessors with work-limited parallelism. In *Proceedings of the International Conference on Real-Time and Network Systems*. 123–128.

COLLETTE, S., CUCU, L. AND GOOSSENS, J. 2008. Integrating job parallelism in real-time scheduling theory. *Inform. Process. Lett. 106*, 5, 180–187.

CUCU, L., AND GOOSSENS, J. 2006. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *Proceedings of the International Conference on Emerging Technologies and Factory Automation*.

CUCU, L., AND GOOSSENS, J. 2007. Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems, In *Proceedings of the 10th Conference on Design, Automation and Test in Europe*.

CUCU, L. 2008. Optimal priority assignment for periodic tasks on unrelated processors, In *Proceedings of the Euromicro Conference on Real-Time Systems, WIP Session*.

DAVARI, S. AND DHALL, S. K. 1986. On a periodic real time task allocation problem. *Annual International Conference on System Sciences*.

DAVIS, R. I. AND BURNS, A. 2009. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *Proceedings of the Real-Time Systems Symposium*.

DHALL, S. K. AND LIU, C. L. 1978. On a real-time scheduling problem. *Oper. Res. 26*, 1, 127–140.

DERTOUZOS, M. L. 1974. Control robotics: The procedural control of physical processes. In *Proceedings of the International Federation for Information Processing Working Conference on Data Semantics*. 807–813.

DERTOUZOS, M. L. AND MOK, A. K. 1989. Multiprocessor scheduling in a hard real-time environment. *IEEE Trans. Softw. Eng. 15*, 12, 1497–1506.

DEVI, U. C. LEONTYEV, H. AND ANDERSON, J. 2006. Efficient synchronization under global EDF scheduling on multiprocessors. In *Proceedings of the Euromicro conference on Real-Time Systems*. 75–84.

DOWNING, C. 2009. *Avionics*, October, 40.

EDMONDS, J. AND PRUH, K. 2009. Scalably scheduling processes with arbitrary speedup curves. In *Proceedings of the Symposium on Discrete Algorithms*. 685–692.

FISHER, N., BARUAH, S. K., AND BAKER, T. P. 2006. The partitioned scheduling of sporadic tasks according to static priorities. In *Proceedings of the EuroMicro Conference on Real-Time Systems*. 118–127.

FISHER, N. AND BARUAH, S. K. 2006. Global static-priority scheduling of sporadic task systems on multiprocessor platforms. In *Proceedings of the International Conference on Parallel and Distributed Computing and Systems*.

FISHER, N. AND BARUAH, S. K. 2007. The global feasibility and schedulability of general task models on multiprocessor platforms. In *Proceedings of the Euromicro Conference on Real-Time Systems*. 51–60.

FISHER, N. 2007. The multiprocessor real-time scheduling of general task systems. Ph.D. dissertation. Department of Computer Science, The University of North Carolina at Chapel Hill, Chapel Hill, NC.

FUNAOKA, K., KATO, S., AND YAMASAKI, N. 2008. Work-conserving optimal real-time scheduling on multiprocessors. In *Proceedings of the Euromicro Conference on Real-Time Systems*. 13–22.

FUNK, S., GOOSSENS, J., AND BARUAH, S. K. 2001. On-line scheduling on uniform multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium*. 183–192.

FUNK, S. AND NADADUR, V. 2009. LRE-TL: An optimal multiprocessor algorithm for sporadic task sets. In *Proceedings of the Real-Time Networks and Systems Conference.* 159–168.

Gai, P., Lipari, G., and Di Natale, M. 2001. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proceedings of the Real-Time Systems Symposium*. 73–83.

Gai, P., Di Natale, M., Lipari, G., Ferrari, A., Gabellini, C., and Marceca, P. 2003. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*.

Garey, M. and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY.

Graham, R. L. 1972. Bounds on multiprocessor scheduling anomalies and related packing problem. In *Proceedings of the AFIPS Spring Joint Computer Conference*. 205–217.

Goossens, J., Funk, S., and Baruah, S. K. 2003. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Syst. 25*, 2–3, 187–205.

Guan, N., Stigge, M., Yi, W., and Yu, G. 2009. New response time bounds for fixed priority multiprocessor scheduling. In *Proceedings of the Real-Time Systems Symposium*.

Holman, P. and Anderson, J. H. 2005. Adapting Pfair scheduling for symmetric multiprocessors. *J. Embed. Comput. 1*, 4, 543–564.

Hong, K. S. and Leung, J. 1988. On-line scheduling of real-time tasks. In *Proceedings of the Real-Time Systems Symposium*. 244–250.

Hong, K. S. and Leung, J.Y.-T. 1992. On-line scheduling of real-time tasks. *IEEE Trans. Comp. 41*, 1326–1331.

Horn, W. A. 1974. Some simple scheduling algorithms. *Naval Res. Logist. Quart. 21*, 177–185.

Ha, R. and Liu, J. W.-S. 1994. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the International Conference on Distributed Computing Systems*. 162–171.

Kalyanasundaram, B. and Pruhs, K. 1995. Speed is as powerful as clairvoyance. In *Proceedings of the Symposium on Foundations of Computer Science*. 214–221.

Kato, S. and Yamasaki, N. 2007. Real-time scheduling with task splitting on multiprocessors. In *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications*. 441–450.

Kato, S. and Yamasaki, N. 2008a. Portioned EDF-based scheduling on multiprocessors. In *Proceedings of the International Conference on Embedded Software*. 139–148.

Kato, S. and Yamasaki, N. 2008b. Portioned static-priority scheduling on multiprocessors. In *Proceedings of the International Parallel and Distributed Processing Symposium*.

Kato, S. and Yamasaki, N. 2008c. Global EDF-based scheduling with efficient priority promotion. In *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications*. 197–206.

Kato, S. and Yamasaki, N. 2009. Semi-partitioned fixed-priority scheduling on multiprocessors. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*. 23–32.

Kato, S., Yamasaki, N., and Ishikawa. Y. 2009. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *Proceedings of the Euromicro Conference on Real-Time Systems*. 249–258.

Khemka, A. and Shyamasundar, R. K. 1997. An optimal multiprocessor real-time scheduling algorithm. *J. Paral. Distrib. Comp. 43*, 1, 37–45.

Lakshmanan, K., Rajkumar, R., and Lehoczky, J. 2009. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *Proceedings of the Euromicro Conference on Real-Time Systems*. 239–248.

Lammers, D. 2004. Intel cancels Tejas, moves to dual-core designs. *EETimes*, May 7.

Lauzac, S., Melhem, R., and Mosse, D. 1998. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *Proceedings of the EuroMicro Workshop on Real-Time Systems*. 188–195.

Lee, S. K. 1994. On-line multiprocessor scheduling algorithms for real-time tasks. In *Proceedings of the IEEE Region 10's 9th Annual International Conference*. 607–611.

Lee, W. Y., Hong, S. J., and Kim, J. 2003. On-line scheduling of scalable real-time tasks on multiprocessor systems. *J. Parall. Distrib. Comp. 63*, 12, 1315–1324.

Lehoczky, J. 1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the Real-Time Systems Symposium*. 201–209.

Leteinturier P. 2007. Multi-core processors: driving the evolution of automotive electronics architectures. *Embedded.com*, September 16.

Leontyev, H. and Anderson, J. H. 2008. Hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the Euromicro Conference on Real-Time Systems*. 191–200.

Lin, X., Lu, Y., Deogun, J., and Goddard, S. 2007. Real-time divisible load scheduling for cluster computing. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*. 303–314.

LIU, C. L. 1969. Scheduling algorithms for multiprocessors in a hard real-time environment. In *JPL Space Programs Summary*, vol. 37-60. JPL, Pasadena, CA, 28–31.

LIU, C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20*, 1, 46–61.

LIU, J. W. S. 2000. *Real Time Systems.* Prentice Hall, Englewood Cliffs, NJ.

LEUNG, J. Y.-T. AND WHITEHEAD, J. 1982. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Perform. Eval. 2*, 237–250.

LOPEZ, J. M., GARCIA, M., DIAZ, J., AND GARCIA, D. F. 2000. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proceedings of the Euromicro Conference on Real-time Systems*. 25–33.

LOPEZ, J. M., DIAZ, J. L., GARCIA, M., AND GARCIA, D. F. 2003. Utilization bounds for multiprocessor RM scheduling. *Real-Time Syst. 24*, 1, 5–28.

LOPEZ, J. M., DIAZ, J. L., AND GARCIA, D. F. 2004a. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. *IEEE Trans. Parall. Distrib. Syst. 15*, 7, 642–653.

LOPEZ, J. M., GARCIA, M., DIAZ, J. L., AND GARCIA, D. F. 2004b. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Syst. 28*, 1, 39–68.

LUNDBERG, L., 2002. Analyzing fixed-priority global multiprocessor scheduling. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*.

LUNDBERG, L. AND LENNERSTAD, H. 2007. Guaranteeing response times for aperiodic tasks in global multiprocessor scheduling. *Real-Time Syst. 35*, 135–151.

LUNDBERG, L. AND LENNERSTAD, H. 2008. Slack-based global multiprocessor scheduling of aperiodic tasks in parallel embedded real-time systems. In *Proceedings of the International Conference on Computer Systems and Applications*. 465–472.

MELLOR-CRUMMEY, J. AND SCOTT, M. 1991. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans. Comp. Syst. 9*, 1, 21–65.

OH, Y. AND SON, S. H. 1993. Tight performance bounds of heuristics for a real-time scheduling problem. Tech. rep. CS-93-24. Department of Computer Science, University of Virginia, Charlottesville, VA.

OH, Y. AND SON, S. H. 1995. Allocating fixed priority periodic tasks on multiprocessor systems. *Real-Time Syst. 9*, 3, 207–239.

OH, D. I. AND BAKER, T. P. 1998. Utilization bounds for *N*-processor rate monotone scheduling with stable processor assignment. *Real-Time Syst. 15*, 2, 183–193.

PARK, M., HAN, S., KIM, H., CHO, S., AND CHO, Y. 2005. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. *IEICE Trans. Inform. Syst. E88-D*, 3, 658–661.

PHILLIPS, C. A., STEIN, C., TORNG, E., AND WEIN, J. 1997. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the ACM Symposium on theory of Computing*.

PIAO, X., HAN, S., KIM, H., PARK, M., CHO, Y., AND CHO, S. 2006. Predictability of earliest deadline zero laxity algorithm for multiprocessor real time systems. In *Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*.

PROARTIS. 2009. Probabilistic analysable real-time systems. EU FP-7-ICT-2009-4 Project proposal.

QUIÑONES, E., BERGER, E., BERNAT, G., AND CAZORLA, F. 2009. Using randomised caches in real-time systems. In *Proceedings of the Euromicro Conference on Real-Time Systems.* 129–138.

RAJKUMAR, R., SHA, L., AND LEHOCZKY, J. P. 1988. Real-time synchronization protocols for multiprocessors. In *Proceedings of the Real Time Systems Symposium*. 259–269.

ROBERTAZZI, T. G. 2003. Ten reasons to use divisible load theory. *Computer 36*, 5, 63–68.

ROTHVOSS, T. 2009. On the computational complexity of periodic scheduling. Ph.D. dissertation. Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland.

SAEZ, S., VILA, J., AND CRESPO. A. 1998. Using exact feasibility tests for allocating real-time tasks in multiprocessor systems. In *Proceedings of the Euromicro Workshop on Real-Time Systems*. 53–60.

SHA, L., RAJKUMAR, R., AND LEHOCZKY, J. P. 1990. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comp. 39*, 9, 1175–1185.

SHA, L., ABDELZAHER, T., ARZEN, K.-E., CERVIN, A., BAKER, T. P., BURNS, A., BUTTAZZO, G., CACCAMO, M., LEHOCZKY, J. P., AND MOK, A. K. 2004. Real time scheduling theory: A historical perspective. *Real-Time Syst. 28*, 2/3, 101–155.

SHIN, I., EASWARAN, A., AND LEE, I. 2008. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proceedings of the Euromicro Conference on Real-Time Systems*. 181–190.

SIMPSON, H. R. 1990. Four-slot fully asynchronous communication mechanism. *IEE Proc. 137 Part E*, 1, 17–30.

SRINIVASAN, A. AND BARUAH, S. K. 2002. Deadline-based scheduling of periodic task systems on multiprocessors. *Inform. Process. Lett. 84*, 93–98.

VEERAVALLI, B., GHOSE, D., AND ROBERTAZZI, T. G. 2003. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Comp. 6*, 1, 7–17.

WILHELM, R., ENGBLOM, J., ERMEDAHL, A., HOLSTI, N., THESING, S., WHALLEY, D., BERNAT, G., FERDINAND, C., HECKMANN, R., MITRA, T., MUELLER, F., PUAUT, I., PUSCHNER, P., STASCHULAT, J., AND STENSTROM, P. 2008. The worst-case execution-time problem overview of methods and survey of tools. *ACM Trans. Embed. Comp. Syst. 7*, 3, 1–53.

ZAPATA, O. U. P AND ALVAREZ, P. M. 2004. EDF and RM multiprocessor scheduling algorithms: Survey and performance evaluation. Report No. CINVESTAV-CS-RTG-02. CINVESTAV-IPN, Sección de Computación, Zacatenco, Mexico.

ZHU, D., MOSSÉ, D., AND MELHEM, R. G. 2003. Multiple-resource periodic scheduling problem: How much fairness is necessary? In *Proceedings of the Real Time Systems Symposium*. 142–151.

Received November 2009; revised January 2010; accepted February 2010