A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems

Robert I. Davis and Alan Burns
Real-Time Systems Research Group, Department of Computer Science,
University of York, YO10 5DD, York (UK)
rob.davis@cs.york.ac.uk, alan.burns@cs.york.ac.uk

Abstract

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 1960's 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.

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 their competition, 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 miniaturisation needed to increase processor clock speeds, as this approach has led to problems with both high power consumption and excessive heat dissipation. Instead, there is now an increasing trend towards using multiprocessor platforms for high-end real-time applications.

A key date in the move towards multiprocessor systems was the 7th May 2004, when Intel cancelled the successor to the Pentium P4 processor called Tejas, due to extremely high power consumption [109].

Dynamic power consumption (the power lost charging and discharging capacitive load) is a dominant factor for chip designs using technology above the 100nm level; however, for sub 100nm technology, transistor leakage current becomes important. This is because the dimensions of the gates and oxide layers are such that the electrical resistance is reduced. The result is that leakage current and hence power dissipation rapidly increases with further miniaturisation. 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 minaturisation and operating frequencies, and instead, use multiple processors on a single chip. On 27th July 2006, two years after cancellation of Tejas, Intel officially released the Core Duo processor. In future, it is expected that 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 11th 2007, has 80 processor cores providing 1 Teraflop performance at 3 GHz.

Multicore processors from other vendors include:

- AMD: Opteron, Phenom, Turion 64, Radeon, and Firestream;
- Analog Devices: Blackfin
- Azul Systems: Vega 1, Vega 2, Vega 3;
- ARM: MPCore;
- Cavium Networks: Octeon;
- Freescale Semiconductor: QorlQ;
- IBM: POWER4, POWER5, POWER6, PowerPC970, Xenon (X-Box 360);
- Intel: Core Duo, Core 2 Duo, Core 2 Quad, Core i3, i5, i7, i9 family, Itanium 2, Pentium D, Pentium Dual-Core, Polaris (teraflops research chip), Xeon.
- Nvidia: GeForce 9, GeForce 200, Tesla;
- NXP Nexperia;
- Sun Microsystems: MAJC 5200, UltraSPARC IV, UltraSPARC T1, UltraSPARC T2;
- Texas Instruments: TMS320C80 MVP;
- Tilera: TILE64;

- XMOS: XS-G4.

1.1. Multiprocessor real-time systems and scheduling

Systems are referred to as real-time when their correct behaviour 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 systems 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 analysing the worst-case behaviour of the application under a given scheduling policy, thus providing proof, subject to a set of assumptions about application behaviour, 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 1960's and early 1970's with significant research effort and advances made in the 1980's and 1990's. The interested reader is referred to [16] and [135] which provide an historical account 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 text books [63], [65], [118], and successfully transferred into industrial practice.

Multiprocessor real-time scheduling theory also has it origins in the late 1960's and early 1970's. In 1969, Liu [116] 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 [78] in 1978 heavily influenced the course of research in this area for two decades. During the 1980's and 1990's, 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 1997 when Phillips et al. [129] showed that the "Dhall effect" was more of a problem with high utilisation tasks than it was with global scheduling algorithms that there was renewed interest in global scheduling algorithms.

In the late 1990's silicon vendors such as IBM, and AMD began research into the development of multicore processors, with IBM releasing the first non-embedded dual-core processor, the POWER4 in 2001.

In the late 1990's, the trend away from increasing processing capacity via ever higher clock speeds, towards increasing performance via multiple processor cores became evident to the real-time systems research community. This resulted in significant research effort being focussed on the problem of real-time multiprocessor scheduling. While markedly more papers 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 paper presents a survey of multiprocessor real-time scheduling algorithms and schedulability analysis techniques, from the origins of the field in the late 1960's up to the latest research published in 2009.

The aim of the survey is to provide a classification of existing research, providing both a perspective on the area, and identifying significant open issues, and future research directions.

1.2. Organisation

The remainder of the paper is organised 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 (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 concludes with the identification of key open issues in the field.

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

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* (or *identical*) 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 ($\tau_1..\tau_n$). 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 inter-arrival 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.

Intra-task 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 characterised by: its relative *deadline* D_i, *worst-case execution time* C_i, and minimum inter-arrival time or *period* T_i. The *utilisation* u_i, of task τ_i is given by C_i / T_i. The utilisation u_{sum} of a taskset is the sum of the utilisations 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, these are:

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 pre-emption and subsequent resumption, a job may, in the case of global scheduling, migrate from one processor to another. The cost of pre-emption, migration, and the run-time 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*: on which processor a task should execute.
2. The *priority problem:* when, and in what order with respect to jobs of other tasks, should each job 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 by Carpenter et al [66]).

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 focussed on models where arbitrary migration (job-level migration) is permitted, in the remainder of this paper 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:

1. *Pre-emptive*: tasks can be pre-empted by a higher priority task at any time.
2. *Non-pre-emptive*: once a task starts executing, it will not be pre-empted and will therefore execute until completion.
3. *Co-operative*: tasks may only be pre-empted at defined scheduling points within their execution. Effectively, execution of a task consists of a series of non-pre-emptable sections.

In this survey, we focus on pre-emptive 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* [29] [30] 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 an interval $[0, t)$ and also has to complete in that interval.

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

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

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

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 condition for taskset feasibility [35]:

$$load(\tau) \le m \qquad (3)$$

where m is the number of processors.

2.6. Notation

For ease of reference, Table 1 provides a summary of the notation used in the rest of the paper.

This notation has been chosen to reflect common usage. Standardising on a common notation such as this would ease communication of results among the research community.

Table 1: 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
R_i	Worst-case response time of task τ_i
$H(\tau)$	Hyperperiod of the taskset
T_i	Minimum inter-arrival time of task τ_i
u_i	Utilisation of task τ_i
$u_{\max}$	Max. utilisation of any task in the taskset.
u_{sum}	Taskset utilisation.
δ_i	Density of task τ_i, $\delta_i = C_i / \min(D_i, T_i)$.
$\delta_{\max}$	Max. density of any task in the taskset.
δ_{sum}	Taskset density (sum of task densities).
$load(\tau)$	Processor load of taskset τ
$load(\tau, k)$	Processor load of taskset τ, due to tasks of priority higher than or equal to *k*.
n	Number of tasks
N	Number of jobs (typically in the hyperperiod of the taskset).
m	Number of processors
t	Time
$h(t)$	Processor demand in the interval [0,*t*)
f_A	Speedup factor (resource augmentation factor) for scheduling algorithm *A*.
$M_A(\tau)$	Minimum number of processors needed to schedule taskset τ using scheduling algorithm *A*.
$\Re_A$	Approximation ratio for scheduling algorithm *A*.
U_A	Utilisation upper bound for scheduling algorithm *A*.

3. Performance metrics

In this section, we describe four performance metrics that have been used to compare the effectiveness of different multiprocessor scheduling algorithms / schedulability analyses. These are:

- Utilisation bounds.
- Approximation Ratio.
- Resource Augmentation or Speedup factor.
- Empirical measures, such as the percentage of tasksets that are found to be schedulable.

3.1. Utilisation bounds

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

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

Note that $\Re_A \ge 1$, 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 f [101], 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:

$$f_A = \max_{\forall m, \forall \tau} \left(f(\tau) \right) \quad (5)$$

Note that $f_A \ge 1$, 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 against 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 [54], 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, 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 pre-emptions 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 un-schedulability.

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
- 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, i.e. all of the feasible tasksets.

In 1974, Horn [99] 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.

In 1988, Hong and Leung [97] [98] 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 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. In 1989, this result was extended by Dertouzos and Mok [80] who showed that knowledge of arrival times is necessary for optimality, even if execution times are known.

In 2007, Fisher [86] 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

In 1974, Horn [99] observed that

$$u_{sum} \le m \tag{6}$$

is a necessary and sufficient condition for the feasibility of implicit-deadline periodic tasksets.

For constrained and arbitrary deadline tasksets, the above condition is necessary, but not sufficient. A tighter necessary condition given by Baruah and Fisher in 2005 [35] is:

$$load(\tau) \le m \tag{7}$$

In 2006, Baker and Cirinei [23], 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 an interval $[0, t)$, even though the release time or deadline is not actually within the interval.

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

Baker and Cirinei 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 (see Equations (1) and (2)):

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

In 2006, Cucu and Goossens [73] showed that the taskset hyperperiod $(0, H]$ 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

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

EDF, an exact schedulability test can be obtained by checking if the schedule generated misses any deadlines in $(0, H]$. 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 $(0, H]$ 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.

In 2007, Cucu and Goossens [74] investigated the feasibility 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 showed that the hyperperiod $(0, H]$ is a feasibility interval provided that all previously released jobs are completed by *H*. For asynchronous, periodic task systems, Cucu and Goossens showed that longer intervals are required to prove exact schedulability.

In 2008, Cucu noted that using the feasibility interval $(0, H]$ and checking all *n*! possible task priority orderings, it is in theory possible to determine exact feasibility for periodic tasksets [75] 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.

In 2007, Fisher and Baruah [85] 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 schedulable 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.

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

Fisher and Baruah 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.

In 2007, Fisher and Baruah [40] also derived a sufficient feasibility test for non-migratory (i.e. partitioned) scheduling. This test states that there exists a partitioning of the tasks that is schedulable using EDF, which is an optimal uniprocessor scheduling algorithm, provided that:

$$load(\tau) \le \frac{1}{3}(m-(m-1)\delta_{\max}) \quad (11)$$

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*: Tasksets exist that are schedulable according to algorithm *A*, but not according to algorithm *B* and vice-versa.

In 2004, Carpenter et al. [66] considered the relationships between the nine different classes of multiprocessor scheduling algorithm (the combinations of the three migration-based and the three priority based categories – see Section 2.3)

Carpenter's key comparability results 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 [119] 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 priorities are essential for optimality.

The maximum possible utilisation bounds (applicable to periodic tasksets with implicit-deadlines) are given in Table 2 below, for algorithms in the various classes:

Table 2

Class	Maximum utilisation bound
Global (job-level migration), dynamic priority	m
All other classes	$(m+1)/2$ [8]

4.4. Predictability

In 1994, Ha and Liu [100] 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 [100] proved that all priority driven, i.e. fixed task priority or fixed job priority, pre-emptive scheduling algorithms for multiprocessor systems are predictable. We note that for any dynamic priority scheduling algorithm, it is necessary to consider / prove predictability before the algorithm can be considered useful.

4.5. Sustainability

In 2006, Baruah and Burns [37] 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 inter-arrival times,
(iii) Increasing deadlines.

Similarly, a schedulability test is referred to as *sustainable* if the above 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* [28] if such a modified taskset will always be 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 test is applied.

While EDF and fixed priority scheduling are sustainable algorithms with respect to uniprocessor scheduling of 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 [28] 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 utilisation, 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 un-sustainability.

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, FF (First Fit) and FFDU (First Fit Decreasing Utilisation) allocation [93]. These anomalies also exist for many fixed task priority partitioning algorithms [9].

In 2003, B. Andersson [9] 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.
- Global optimal scheduling (full migration, dynamic priorities) of synchronous periodic tasksets.

The interested reader is referred to Chapter 5 of B. Andersson's thesis [9] for a set of illustrative examples.

4.6.2 Critical instant effect

In 1998, Lauzac et al. [110] 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, the response time of a low priority task is longer if when the task executes, zero or only a few higher 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 task of interest 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.

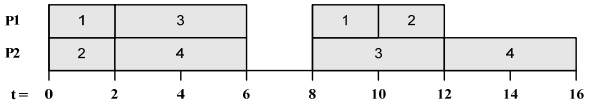

Figure 1: Critical instant effect

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 4 time units in the interval [8, 15), whereas they only occupy both processors for 2 time units in the interval [0, 7).

In his thesis, B. Andersson [9] observes 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 optimal priority assignment algorithm [15] [17] for the uniprocessor case, is not applicable to the multiprocessor case, when schedulability analysis uses exact response times. In 2009, Davis and Burns [77] 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. In [38] Baruah remarks that, *"no finite collection of worst-case job arrival sequences has been identified for the global scheduling of sporadic task systems."* 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, then there is no penalty in terms of migration cost. For example, a job that is started on one processor, then pre-empted 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 / non-migration case. This problem could be mitigated by allowing only task, as opposed to job-level migration, or by non-preemptive execution, although the later could result in significant loss of schedulability due to long non-pre-emptive sections.
- 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 pre-emptive 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 [116].
- Similarly, Deadline Monotonic (DM) priority assignment is optimal for such tasksets with constrained-deadlines [119]. (We note that DM is not optimal for tasksets with arbitrary deadlines [113], or for asynchronous periodic tasksets; however Audsley's priority assignment algorithm is known to be optimal in these cases [15] [17]).

Considering pre-emptive uniprocessor scheduling using fixed job priorities:

- EDF is the optimal scheduling algorithm for sporadic tasksets independent of the deadline constraints [79].

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 [92].

5.1. Implicit-deadline tasksets

Early research into partitioned multiprocessor scheduling by Dhall and Liu [78] in 1978, Davari and Dhall [76] in 1986, Oh and Son [126], [127] in 1993 and 1995, and Burchard et al. [62] in 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), "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, meaning Rate Monotonic (fixed priority) scheduling with Best Fit task allocation.

5.1.1 Approximation Ratio

Table 3 below gives the approximation ratio required for each of these algorithms for periodic tasksets with implicit-deadlines.

Recently in 2009, Rothvoß [132] 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 [62].

Table 3

Algorithm	Approximation Ratio ($\Re_A$)	Ref.
RMNF	2.67	[78]
RMFF	2.33	[126]
RMBF	2.33	[126]
RM-FFDU	5/3	[127]
FFDUF	2	[76]
RMST	$1/(1-u_{\max})$	[62]
RMGT	7/4	[62]
RMMatching	3/2	[132]
EDF-FF	1.7	[92]
EDF-BF	1.7	[92]

Note, $u_{\max}$ is the highest utilisation of any task in the taskset.

While these approximation ratios enable a comparison to be made between the different algorithms, their practical use as a schedulability test is severely limited, as 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 utilisation bounds that can be derived from these approximation ratios are pessimistic [128].

5.1.2 Utilisation bounds

In 2001, B. Andersson et al. [8] showed that for periodic tasksets with implicit-deadlines, the largest worst-case utilisation bound for any partitioning algorithm is:

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

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 utilisation tasks were recognised early on by the research community; leading to a significant thread of research during the 1990's providing utilisation bounds as a function of $u_{\max}$, the highest utilisation of any task in the taskset.

In 1995, Burchard et al [62] provided utilisation 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 favours tasks with utilisation < 1/2:

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

Burchard et al [62] also provided utilisation bounds for the RMGT ("General Tasks") algorithm, which separates tasks into two groups depending on whether their utilisation is above or below 1/3:

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

In 1998, Oh and Baker [128] showed that RM-FFDU has a utilisation bound given by:

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

They also showed that the utilisation bound for any fixed task priority partitioning algorithm is upper bounded by:

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

Lopez et al [121], [122], [123] subsequently generalised the above result for RM-FFDU, and also provided more complex bounds based on the number of tasks n and the value of $u_{\max}$ for RMBF, RMFF, and RMWF.

In 2003, B. Andersson [10] showed that the RBOUND-MP-NFR algorithm has a utilisation bound of:

$$U_{RBOUND-MP-NFR} = m/2 \quad (17)$$

This result shows that a fixed task priority partitioning algorithm exists that is an optimal partitioning approach in the limited sense that its utilisation 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.

In 2000, Lopez et al [120] showed that using EDF, the lowest utilisation bound for any *reasonable*[2] allocation algorithm is given by:

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

and that the highest utilisation bound of any reasonable allocation algorithm is:

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

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

Lopez et al. [120] showed that all reasonable allocation algorithms that order tasks by decreasing utilisation achieve the higher limit, as do EDF-BF and EDF-FF. Further, EDF-WF, but not EDF-WFDU,

[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.

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 utilisation bounds are as large as that of any partitioning algorithm.

We note that for applications with "small" tasks, then RMST and EDF-FF provide reasonably high utilisation bounds. For example, assuming $m = 10$ and $u_{\max} = 0.25$, the utilisation bounds for RMST and EDF-FF are 63% and 82% respectively.

5.2. Constrained and arbitrary deadline tasksets

In 2005, Baruah and Fisher [35] showed that EDF-FFD (decreasing density) is able to schedule any arbitrary-deadline sporadic taskset provided that:

$$\delta_{sum} \le \begin{cases} m - (m-1)\delta_{\max} & \delta_{\max} \le 1/2 \\ m/2 + \delta_{\max} & \delta_{\max} \ge 1/2 \end{cases} \quad (20)$$

The resource augmentation factor for EDF-FFD is however, *not* finite [35].

Baruah and Fisher [35], [36], [39] also developed an algorithm 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:

$$m \ge \left(\frac{2load(\tau) - \delta_{\max}}{1 - \delta_{\max}} \right) \quad (21)$$

For tasksets with arbitrary deadlines, the test becomes:

$$m \ge \frac{load(\tau) - \delta_{\max}}{1 - \delta_{\max}} + \frac{u_{sum} - u_{\max}}{1 - u_{\max}} \quad (22)$$

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.

In 2006, Fisher et al. [83] 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 is able to schedule any sporadic taskset with constrained-deadlines provided that:

$$m \ge \frac{load(\tau) + u_{sum} - \delta_{\max}}{1 - \delta_{\max}} \quad (23)$$

For tasksets with arbitrary deadlines, the test becomes:

$$m \ge \frac{load(\tau) + u_{sum} - \delta_{\max}}{1 - \delta_{\max}} + \frac{u_{sum} - u_{\max}}{1 - u_{\max}} \quad (24)$$

Fisher et al. [83] 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 / pre-emptions when global scheduling is used, this is because the scheduler will only pre-empt a task when there are no processors idle [6].
- Spare capacity created when a task executes for less than its worst-case execution time can be utilised 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 behaviour of the entire system, with all tasks taking worst-case execution times, worst-case phasing occurring etc. is less likely 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 focussed on approaches that permit job-level migration, where a job may be pre-empted 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 in 1978 [78] considered global scheduling of periodic tasksets with implicit deadlines on m processors. They showed that the utilisation bound for global EDF scheduling is $1 + \varepsilon$, for arbitrary small ε. This occurs when there are m tasks with short periods/deadlines and infinitesimal utilisation, and one task with a longer period/deadline and utilisation that approaches 1.

This "Dhall effect" led to a general view that global approaches to multiprocessor scheduling were inferior to partitioned approaches. As a result, throughout the 1980's and early 1990's, the majority of research into

multiprocessor real-time scheduling focussed on partitioned approaches, as described in the previous section.

In 1997, Phillips et al. [129] showed that augmenting a system by increasing processor speed is more effective that 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 schedulable by global EDF).

The resource augmentation results of Phillips et al. [129], along with research by Funk et al. [88] in 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 utilisation. This observation was exploited in much of the subsequent research to provide utilisation bounds that are dependent on the maximum task utilisation $u_{\max}$.

6.1. Global fixed job priority scheduling

6.1.1 Implicit deadline tasksets

In 2001, B. Andersson et al. [8] considered utilisation bounds for periodic tasksets with implicit deadlines. They showed that the maximum utilisation bound for any global fixed job priority algorithm is:

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

In 2002, Srinvasan and Baruah [138] proposed the EDF-US[ς] algorithm that gives the highest priority to tasks with utilisation greater than the threshold ς, with ties broken arbitrarily. Setting the threshold to $m/(2m-1)$ results in a utilisation bound that is independent of $u_{\max}$:

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

In 2003, Goossens et al. [94] derived a utilisation bound for global EDF applicable to periodic tasksets with implicit-deadlines and showed that this bound is tight:

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

Later that year, Baruah and Carpenter [34] showed that this same utilisation bound applies to global EDF scheduling, assuming task level, migration.

Goossens et al. [94] also proposed an algorithm called EDF(k) that assigns the highest priority to the k tasks with the highest utilisation. They showed that a sufficient schedulability condition for EDF(k) is:

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

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

In 2005, Baker [20], [24] showed that setting the threshold used in EDF-US[ς] to 1/2, results in the following utilisation bound which is the maximum possible bound for this class of algorithm [8]:

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

Baker [20] 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 showed that the utilisation bound for EDF($k_{\min}$) is also:

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

Again, this is the maximum possible utilisation 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 utilisation bound given in Equation (27) was extended by Bertogna et al. [48] to the case of sporadic tasksets with constrained deadlines and by Baruah and Baker [25] to the arbitrary-deadline case, giving the following sufficient schedulability test based on task density:

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

Bertogna [49] also adapted the utilisation 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 than the threshold ς. Bertogna showed that a sporadic taskset is schedulable according to EDF-DS[1/2] provided that:

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

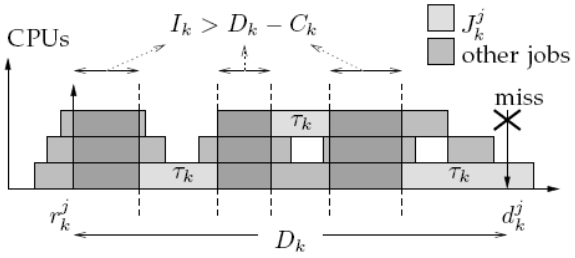

Figure 2: Problem window

In 2003, Baker [19] 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 arrival 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 both 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 un-schedulability 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 in [19] 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 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 pre-emptive global EDF scheduling if for every task τ_k*:*

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

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 11in [19] for a definition of β_i.

In 2005, Baker [20] 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 in [19] is a seminal result which has been built upon by a significant thread of subsequent research.

In 2005, Bertogna et al. [48] showed that the test proposed by Baker [20] (Equation (33)) does not dominate the extended version of test proposed by Goossens et al. [94] (Equation (31)). In fact, the test given by Equation (33) performs relatively poorly when tasks with high individual utilisations are considered. Bertogna et al. [48] 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 summarised as follows:

A constrained-deadline taskset is schedulable under pre-emptive global EDF scheduling if for every task τ_k*, one of the following holds:*

$$\sum_{\forall i \neq k} \min(\beta_k(i), 1-\delta_k) < m(1-\delta_k)$$

or

$$\sum_{\forall i \neq k} \min(\beta_k(i), 1-\delta_k) = m(1-\delta_k)$$

$$\text{and } \exists i \neq k : 0 < \beta_k(i) \leq 1-\delta_k \quad (34)$$

where

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

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 ($n >> m$). This happens because every task is counted as contributing some carry-in interference. Further, these tests tend to perform poorly on tasksets where the task parameters are of different orders of magnitude.

In 2007, Baruah [38] derived a sufficient schedulability test for global EDF scheduling of sporadic tasksets with constrained deadlines. This test uses the same basic approach as Baker [19] 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 in [38] 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.

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

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 pseudo-polynomial in complexity.

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

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

where

$$I_k^{\in} = \sum_{\forall i (m-1) largest} (I''_k(i) - I'_k(i))$$

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

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

Notably, the above test reverts to the processor load-based test ($load(\tau) \le 1$) for uniprocessor systems [29], [30], when $m = 1$.

In 2008, building on their previous work [38], Baruah and Baker [27] 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:

$$load(\tau) \le \mu - (\lceil \mu \rceil - 1)\delta_{\max} \quad (38)$$

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

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

$$f = \frac{2}{3-\sqrt{5}} \approx 2.62 \quad (39)$$

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

Later in 2008, Baruah and Baker [43] extended the results in [27] to sporadic tasksets with arbitrary deadlines, showing that Equation (38) still applies. Baruah and Baker [44] 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 artefact of the analysis used.

In 2008, Bertogna et al. [51], 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.

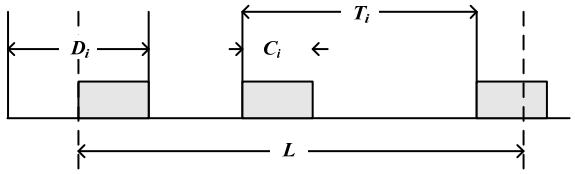

Figure 3: Densest packing for work conserving algorithms

Bertogna et al. [51] showed that $W_i(L)$ is an upper bound on the workload of task τ_i in an interval of length L.

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

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

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

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

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

Bertogna et al. [51] 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:

$$\sum_{i \ne k} \min(\Lambda_k^i(D_k), D_k - C_k + 1) < m(D_k - C_k + 1) \quad (43)$$

where:

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

Bertogna et al. 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 specialised 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 [85], see Section 4.2, Equation (10). This iterative test has complexity that is pseudo polynomial: $O(n^3 D_{\max}^2)$.

In 2007, Baruah and Fisher [40] 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*:

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

where *K* is the largest ratio task deadlines. As *K* may potentially take any value, this test does not have a finite resource augmentation factor.

In 2009, Baker and Baruah [28] showed that the sufficient schedulability tests for global EDF given in [20], [27], and [48] (Equations (33), (34) and (38) are unsustainable with respect to increases in relative deadline, and the test given in [27], 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 [44] improved the sufficient test from [27], Equation (38), making it sustainable. The improved schedulability test is as follows:

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

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

In 2008, Bonifaci et al. [64] 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. [129] 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. [64] and extended by Baruah et al. in [45] 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: global fixed task priority scheduling.
- Global RM scheduling: global FP scheduling using Rate Monotonic priority ordering.
- Global DM scheduling: 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 in 1978 [78] also considered global scheduling of periodic tasksets with implicit deadlines on *m* processors. They showed that the utilisation bound for global RM scheduling is $1+\varepsilon$, for arbitrarily small ε. This occurs when there are *m* tasks with short periods/deadlines and infinitesimal utilisation, and one task with a longer period/deadline and utilisation that approaches 1.

In 2000, B. Andersson and Jonsson [6], 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.

$$k = \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{2m} \quad (46)$$

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

In 2001, B. Andersson et al. [8] showed that any periodic taskset with implicit deadlines can be scheduled using global RM scheduling provided that:

$$u_{\max} \le m/(3m-2) \text{ and } u_{sum} \le m^2/(3m-1) \quad (47)$$

This result, albeit in a weaker form, also appeared in a paper [33] by the co-authors Baruah and Goossens in 2003:

$$u_{\max} \le 1/3 \text{ and } u_{sum} \le m/3 \quad (48)$$

B. Andersson et al. [8] also proposed the RM-US[ς] algorithm that gives the highest priority to tasks with utilisation greater than the threshold ς (with ties broken arbitrarily), and otherwise assigns priorities in RM order. B. Andersson et al. showed that RM-US[$m/(3m-2)$] has a utilisation bound of:

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

In 2002, Lundberg [124] showed that that setting the threshold used in RM-US[ς] to 0.375 results in the following utilisation bound which is the maximum possible bound for this algorithm:

$$U_{RM-US[0.375]} \approx 0.375m \quad (50)$$

In 2003, B. Andersson and Jonsson [10] showed that for periodic tasksets with implicit deadlines, the maximum utilisation 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:

$$U_{OPT} \le (\sqrt{2}-1)m \approx 0.41m \quad (51)$$

In 2005, Bertogna et al. [47] tightened the bound for global RM scheduling to:

$$u_{sum} \le \frac{m}{2}(1-u_{\max}) + u_{\max} \quad (52)$$

In 2008, B. Andersson [14] 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. B. Andersson showed that SM-US[$2/(3+\sqrt{5})$] has a utilisation bound of:

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

for sporadic tasksets with implicit deadlines.

6.2.2 Constrained and arbitrary deadline tasksets

In 2000, B. Andersson and Jonsson [7] 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.

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

In 2003, Baker [19], [21] applied the same general strategy described in section 6.1.2 for global EDF to global FP scheduling of sporadic tasksets with

constrained deadlines.

In 2005, Bertogna [47] proved the following density bound for global DM scheduling of sporadic tasksets with constrained deadlines:

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

Bertogna et al. [47] 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:

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

Where ψ is the number of 'privileged' tasks with density higher than the threshold, and $\delta^{(m)}$ is the density of the *m*th highest density task.

Bertogna et al. [47] proved the following sufficient test for DM-DS[1/3]:

$$\delta_{sum} \le \frac{m+1}{3} \quad (57)$$

Bertogna et al. [47] also proposed the following 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.

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

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

or

$$\sum_{i<k} \min(\beta_k(i), 1-\delta_k) = m(1-\delta_k)$$

$$\text{and } \exists i \ne k : 0 < \beta_k(i) \le 1-\delta_k \quad (58)$$

where

$$\beta_k(i) = \frac{N_{i,k}C_i + \min(C_i, (D_k - N_{i,k}T_i + D_i - C_i)_0)}{D_k}$$

$$N_{i,k} = \left\lfloor \frac{D_k - C_i}{T_i} \right\rfloor + 1$$

In 2006, Fisher and Baruah [84], derived a sufficient test for global DM scheduling of sporadic tasksets with arbitrary deadlines, under the assumption that *intra-task parallelism*, where jobs of the same task can execute in parallel on different processors, is permitted; while *inter-job parallelism* is not.

This sufficient test for each task τ_k is as follows:

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

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.

In 2007, Baruah [46] derived an alternative sufficient test for global DM scheduling of sporadic tasksets with constrained deadlines using a similar approach to [84], but limiting the amount of carry-in execution in a different way. This sufficient test for each task τ_k is as follows:

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

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:

$$load(k) \le \frac{1}{2}(m-(m-1)C_k/D_k)(1-\delta_{\max}(k)) \quad (61)$$

where $\delta_{\max}(k)$ is the maximum density of any of the *k* highest priority tasks. Baruah 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 non-optimality of global DM scheduling and the sufficiency of the test.

In 2008, Fisher and Baruah [41] showed that the result derived in [84], Equation (59), also applies to systems where intra-task 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 $(4-1/m)$.

In 2008, Bertogna et al. [51] specialised 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:

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

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. [51] 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 [50] 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 in the following fixed point iteration:

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

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:

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

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

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

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:

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

In 2009, Guan et al. [95] extended the response time analysis of Bertogna and Cirinei [50], limiting the amount of 'carry-in' interference using ideas from [38]. An important observation following from the analysis of Guan et al. [95] 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. [95] showed that if task τ_i does not have a carry-in job, then the interference is given by:

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

where:

$$W_i^{NC}(L) = N_i^{NC}(L)C_i + \min(C_i, L - N_i^{NC}(L)T_i) \quad (70)$$

and

$$N_i^{NC}(L) = \left\lfloor \frac{L}{T_i} \right\rfloor \quad (71)$$

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

$$I_i^{DIFF}(R_k^{UB}) = I_i^{CI}(R_k^{UB}) - I_i^{NC}(R_k^{UB}) \quad (72)$$

Using this result, Guan et al. [95] improved upon the response time test of Bertogna and Cirinei [50] as follows:

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m}\left(\sum_{\forall i \in hp(k)} I_i^{NC}(R_k^{UB}) + \sum_{i \in Max(k,m-1)} I_i^{DIFF}(R_k^{UB}) \right) \right\rfloor \quad (73)$$

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. [95] (Equation (73)) dominates the response time test of Bertogna and Cirinei [50] (Equation (63)) which in turn dominates the deadline based test of Bertogna et al. [51].

Guan et al. [95] also extended their response time test to tasksets with arbitrary deadlines.

In 2009, Davis and Burns [77] showed that Audsley's optimal priority assignment algorithm [15] [17] is applicable to some sufficient tests for global FP scheduling, including those of B. Andersson and Jonsson [7] (Equation (54)) and Bertogna et al. [51] (Equation (62)). Davis and Burns also extended the TkC priority assignment policy to sporadic tasksets to form "DkC" priority assignment, which orders task priorities based on their deadlines less some constant (given by Equation (46) times 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, LLREF). However, it is known that there are no optimal online (non-clairvoyant) algorithms for the pre-emptive scheduling of sporadic tasksets on multiprocessors [86].

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 pre-emption and migration.

6.3.1 Proportionate fairness algorithms

The *Proportionate Fair* (Pfair) algorithm was introduced by Baruah et al. in 1996 [32]. 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 utilisation (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. [32] showed that the Pfair algorithm is optimal for periodic tasksets with implicit deadlines, with a utilisation bound of:

$$U_{PFAIR} = m \quad (74)$$

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

A number of variants on the Pfair approach have been introduced, including ERFair [2], PD [31], and PD^2

[4]. 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 is between -1 and +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 [31], and PD^2 [4] improve on the efficiency of Pfair by separating tasks into groups of *heavy* ($u_i > 0.5$) and *light* tasks.

In 2000, J. Anderson and Srinivasan [3] extended the PFair approach to sporadic tasksets, showing that the EPDF (*earliest pseudo-deadline first*) algorithm, a variant of PD, is optimal for sporadic tasksets with implicit deadlines executing on 2 processors, but is not optimal for more than 2 processors.

In 2003, Zhu et al. [140] introduced the B*oundary Fair* (BF) algorithm. Zhu et al. recognised 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. [140] 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.

In 2005, Holman and J. Anderson [96] implemented Pfair scheduling on a symmetric multiprocessor. They found that the synchronised re-scheduling of all processors every time quanta caused significant bus contention due to data being re-loaded into cache. To address this problem, Holman and J. Anderson [96] 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 SA

In 1997, Khemka and Shyamasundar [107] developed an optimal algorithm for periodic tasksets with implicit-deadlines called SA This algorithm takes at most $O(n+m+1)H(\tau)/GCD(\tau)$ operations to build a schedule, where $H(\tau)$ is the least common multiple of task periods and $GCD(\tau)$ is the greatest common divisor of the task periods. We note that as with Pfair, the number of task pre-emptions with SA can be prohibitively large.

6.3.3 LLREF

In 2006, Cho et al. [69] 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, i.e. task releases, and co-incident 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, i.e. $t_f^k u_i$, 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 non-running 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. [69] 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.

In 2008, Funaoka et al. [87] extended the LLREF approach, apportioning processing time that would otherwise be unused among the tasks, and re-apportioning processing time when a task completes earlier than expected, thus creating a work-conserving algorithm. Funaoka et al. [87] showed that for taskset utilisations below 100% this approach results in significantly fewer pre-emptions than LLREF.

In 2009, Funk and Nadadur [89] extended the LLREF approach, forming the LRE-TL algorithm. The key observation of Funk and Nadadur 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 maximum number of migrations per section, compared to LLREF. Funk and Nadadur also showed how the LRE-TL algorithm could be applied to sporadic tasksets and proved that it is optimal (utilisation bound of 100%) for sporadic tasksets with implicit deadlines.

6.3.4 EDZL

In 1994, Lee [111] 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.

In 2007, Cirinei and Baker [70] showed that EDZL

is *predictable* in the sense defined by Ha and Liu [100], (see Section 4.4). Cirinei and Baker provided the following sufficient schedulability test for EDZL:

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:

$$\sum_{\forall i \neq k} \beta_k^i \geq m(1-\lambda_k) \quad (75)$$

Where, $\lambda_k = C_i / \Delta_k$ and $\Delta_k = \min(D_i, T_i)$ and

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

In 2008, Baker et al [26] refined the sufficient test for EDZL, replacing Equation (75) with:

$$\sum_{\forall i \neq k} \min(\beta_k^i, 1-\lambda_k) \geq m(1-\lambda_k) \quad (76)$$

They also gave an iterative sufficient test for EDZL based on the approach taken by Bertogna et al. in [51] for work conserving algorithms and EDF. The tests given by Equations (75) and (76) suffer from an over-estimation of the amount of carry-in interference, particularly for tasksets with cardinality $n >> m$. The iterative test of Baker et al [26] 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 in [26] shows that this iterative test for EDZL outperforms previous tests given in [70] and (as expected) similar tests for global EDF.

Also in 2008, Chao et al [67] showed that the utilisation bound for EDZL, assuming tasksets with implicit deadlines and large *m*, is:

$$U_{EDZL} \leq m(1-1/e) \approx 0.63m \quad (77)$$

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 / non-migration 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 utilisation bound is just 50% of the total processing capacity.

In this section we outline recent research into hybrid approaches which combines elements of both partitioned and global scheduling.

7.1. Semi-partitioned approaches

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

In 2006, B. Andersson and Tovar [11] 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. Andresson and Tovar showed that the utilisation bound for EKG depends on the parameter *k*, used to control division of tasks into groups of heavy and light tasks. The utilisation bound for EKG is given by:

$$U_{EKG} = \begin{cases} k/k+1 & k < m \\ 1 & k \end{cases} \quad (78)$$

Hence the utilisation bound is 100% for $k = m$. Further, the average number of pre-emptions per job over the hyperperiod is bounded by 2*k*. Thus as suggested by B. Andersson and Tovar, choosing a value of $k = 2$, gives a utilisation bound of 66% and at most an average of 4 pre-emptions per job.

In 2008, B. Andersson and Bletsas [13] 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 executed by processor *p*-1 and one executed by processor *p*+1. B. Andersson et al. [12] later extended this approach to tasksets with arbitrary deadlines. They showed that first-fit and next-fit were not good allocation strategies when task splitting is employed. Instead, they ordered tasks by decreasing relative deadline and tried to fit all tasks on the first processor before then choosing the remaining task with the shortest relative deadline to be split. At run-time, 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 pre-emptions is increased if the time slots are short.

In the implicit deadline case, B. Andersson and Bletsas [13] showed that this approach has a utilisation bound of:

$$U = 4(\sqrt{\delta(\delta+1)} - \delta) - 1 \quad (79)$$

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

$$3\delta \lceil t / T_{\min} \rceil + 2 \quad (80)$$

In 2009, Bletsas and B. Andersson [57] 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 organised 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 B. Andersson showed that this method has a utilisation bound of at least 66.6% for tasksets with implicit deadlines, and that the number of additional pre-emption, above those caused by task arrivals is given by:

$$\left(2m + \left\lceil \frac{m}{3} \right\rceil\right)\left\lceil \frac{t}{S} \right\rceil \quad (81)$$

where t is the length of the time interval, and S is the minimum period of any task on the processor considered. This number of additional pre-emptions compares favourably with that given by Equation (80) even when $\delta = 1$.

In 2007, Kato and Yamasaki [102] 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 utilisation bound of 50%.

In 2008, Kato and Yamasaki [103], 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 utilisation 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 [103] showed that EDDP has a utilisation 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 [104] also extended this approach to fixed task priority scheduling, showing that the RMDP algorithm has a utilisation bound of 50%.

In 2009, Kato et al. [105] developed a semi-partitioning 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 run-time, 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 et al. [105] showed that DM-PM has a utilisation bound of 50% for tasksets with implicit deadlines. Subsequently, Kato et al. [106] extended the same basic approach to EDF scheduling; forming the EDF-WM algorithm (EDF with Window constrained Migration).

In 2009, Lakshmanan et al. [108] developed a semi-partitioning 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 pre-emptive 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 for the part of the task split on to another processor to execute.

Lakshmanan et al. [108] showed that for any task allocation PDMS_HPTS has a utilisation bound of 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 utilisation bound of 69.3% if the maximum utilisation of any individual task is no greater than 0.41. Notably, this is the same as the Liu and Layland bound [117] for single processor systems, without the restriction on individual task utilisation.

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 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.

In 2008, Shin et al. [136] 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. develop 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 pre-emptions 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.

In 2008, Leontyev and J. Anderson [114], 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 utilised processors and at most one processor which is partially utilised. This partial bandwidth is provided by a periodic server. Leontyev and J. Anderson showed how the tardiness of soft real-time tasks can be bounded in this model, without any loss of utilisation. Utilisation loss does occur when hard-real-time tasks are included; however, this loss is shown to be small provided that the utilisation of hard real-time tasks represents 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 survey research lifting this assumption of independence and therefore allowing tasks to share resources that have to be accessed in mutual exclusion.

We note that as an alternative to mechanisms that support mutual exclusion, there are non-blocking solutions to the specific problem of single-writer, single-reader communication which can support asynchronous access by tasks on multiprocessors for example, Simpson's four-slot mechanism [137], which preserves independence of execution. This mechanism does however require memory space for four copies of the data. For more complex cases with multiple writers, and access to other types of shared object, e.g. registers in hardware peripherals, then mutual exclusion is required.

In uniprocessor systems, the Stack Resource Policy (SRP) [18] and Priority Ceiling Protocol (PCP) [134] 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 built on these uniprocessor protocols.

8.1. Partitioned scheduling

In 1988 Rajkumar et al [130] 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 run-time when a task attempts to access a locked global resource, it is suspended, and waits in a FIFO 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 utilisation bound of Liu and Layland. The blocking factor is made up of five different components, which are summarised in [91].

In 1994 Chen et al. [68] 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.

In 2001, Gai et al [90] introduced the MSRP protocol based on SRP [18]. 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 pre-emptable. This behaviour 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 analysed 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.

In 2003, a study performed by Gai et al. [91] 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

In 2006, Devi et al [81], considered the problem of accessing mutually exclusive shared resources under global EDF scheduling. They suggested two simple approaches for short non-nested accesses to shared data structures: *Spin-based queue locks* [125] and lock-free synchronisation.

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 [81], the spin queue grants access to resources in FIFO order, further access to each resource is non-pre-emptable; 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. [81] 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 synchronisation using retries. The performance evaluation reported in [81] suggests that the total overheads of spin-based queue locks are significantly less than that of lock-free synchronisation.

In 2007, Block et al. [58] 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 non-preemptable job when it is release 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 non-preemptable 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 optimises the simple case of non-nested access to short resources. In [58], Block et al. 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.

In 2008, Brandenburg et al. [61] examined the relative performance of blocking and non-blocking approaches to accessing shared resources. The blocking approaches used FMLP and considered both spinning, i.e. busy-waiting, and suspending. The non-blocking approaches considered were *lock-freedom* and *wait-freedom*. Brandenburg et al. concluded that non-blocking approaches are preferable for small and simple resource objects and that for more complex resource objects with longer access times wait-free or spin-based algorithms are generally preferable. Suspension based algorithms were 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 regarding approximation ratios, utilisation 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 being highly effective for the majority of realistic cases. While utilisation / 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 utilisation, range of task periods, distribution of task deadlines, distribution of individual task utilisations, can be varied to examine the performance of the algorithms and their schedulability tests over a range of different credible scenarios.

In 2005, Baker [22] made an empirical comparison between the best global EDF, and partitioned EDF scheduling algorithms available at that time. 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.

In 2007, considering global scheduling algorithms, in the simulation chapter of his thesis, Bertogna [49] 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 6.3.4), known at the time. Since real-time system designers are interested in provable schedulability, Bertogna argues 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.

In 2009, Bertogna [52] investigated the performance of the following schedulability tests for global EDF:

- Goossens et al. [94] (GFB) density-based test;
- Baker [20] (BAK);
- Baruah [38] (BAR);
- Baruah and Baker [44], (LOAD) processor load based test;
- Bertogna et al. [48] (BCL);
- Bertogna and Cirinei [50] (RTA) response time analysis test;
- Baruah et al. [45] (FF-DBF) speedup optimal test.

Bertogna showed that of these tests, the RTA test [50] 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 also sequentially applied the RTA test, the BAR test and the FF-DBF test to form a composite test (COMP). This test utilises 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.

In 2009, Davis and Burns [77] showed that, in global FP scheduling, the number of randomly generated tasksets deemed schedulable using the schedulability tests of Bertogna et al. [51] is significantly increased by using Audsley's optimal priority assignment policy [15] [17] 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 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 [77] also showed how the UUnifast method of taskset generation [54], 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 [50], [52] while valid in comparing the performance of different schedulability tests, suffers from a problem of confounding variables: As taskset utilisation is increased, so 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

In 2008, Brandenburg et al. [60] measured the performance of various scheduling algorithms and their overheads on a LITMUS test-bed using a Sun UltraSPARC Niagra multicore platform with 32 logical processors (actually 4 hardware threads on each of 8 CPUs). Brandenburg et al. 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 utilisations, then staggered Pfair was best.

For soft real-time tasksets, partitioned EDF was again effective unless the tasks had high individual utilisations >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

[78], there are still significant and fundamental research challenges that remain.

Global, clustered, and semi-partitioned 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 utilisation: Fixed job-priority and partitioning algorithms are in the worst-case capable of utilising only 50% of the available processing resource. While global dynamic algorithms can in some cases utilise 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 summarised in Section 7.

Ineffective schedulability tests: For the sporadic task model, empirical studies have shown that there is a large gap that 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 [38].

Consideration of overheads: Advanced hardware features, such as cache architectures, have a large impact on the cost of migration (at the task and job level). Recent experimental implementations [60] on multiprocessor platforms show 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 addresses 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. [71], [72] considers 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 [82] 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 on-line scheduling methods covering scalable tasks [112], and the application of Divisible Load Theory [53], [131], [139], [115] 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. [55], [56] models 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 [18] 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 indicates that spin-based approaches appear to be preferable to suspension-based methods; however non-blocking 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:

- Worst-case Execution Time (WCET) analysis;
- Network / bus scheduling;
- Memory architectures;
- Uniform and heterogeneous processors;
- Operating Systems;
- Power consumption and dissipation;
- Scheduling tasks with soft real-time constraints.
- Non-real-time issues such as load balancing.

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 over-specified 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 of 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.

11.1. Acknowledgements

The authors would like to thanks Sanjoy Baruah for his comments and suggestions on an earlier draft.

This work has been funded in part by the EU FP7 projects eMuCo and Jeopard, the EU FP7 ArtistDesign network of excellence, and the EPSRC project Tempo.

References

[1] J. Anderson, S. Ramamurthy, K. Jeffay, Real-Time Computing with Lock-Free Shared Objects ACM Transactions on Computer Systems, Volume 15, Number 2, pp. 134-165, May 1997.

[2] J. Anderson, A. Srinivasan. Early-release fair scheduling. In Proceedings Euromicro Conference on Real-Time Systems, June 2000.

[3] J. Anderson, A. Srinivasan. “Pfair scheduling: Beyond periodic task systems”. In Proceedings of the 7th International Workshop on Real-Time Computing Systems and Applications, December 2000.

[4] J. Anderson, A. Srinivasan. “Mixed pfair/erfair scheduling of asynchronous periodic tasks”. In Proceedings of the 13th Euromicro Conference on Real-Time Systems, June 2001.

[5] J.H. Anderson, A. Srinivasan, “Mixed Pfair/ERfair scheduling of asynchronous periodic tasks”, Proceedings of the EuroMicro Conference on Real-Time Systems (Delft, The Netherlands), IEEE Computer Society Press, June 2001.

[6] B. Andersson, J. Jonsson, “Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition”, Proceedings of the International Conference on Real-Time Computing Systems and Applications, Cheju Island, Korea 2000.

[7] B. Andersson, J. Jonsson, “Some insights on fixed-priority pre-emptive non-partitioned multiprocessor scheduling”. In Proceedings Real-Time Systems Symposium– Work-in-Progress Session, Nov. 2000.

[8] B. Andersson, S. Baruah, J. Jonsson. Static-priority scheduling on multiprocessors. In Proc. 22nd IEEE Real-Time Systems Symposium, pages 193–202, London, UK, Dec. 2001.

[9] B. Andersson, "Static-priority scheduling on multiprocessors",Ph.D. thesis. Chalmers University of Technology, 2003.

[10] B. Andersson, J. Jonsson, "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 (ECRTS), 2003.

[11] B. Andersson, E. Tovar, “Multiprocessor Scheduling with Few Preemptions”. In Proceedings International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) August, 2006.

[12] B. Andersson, K. Bletsas, S.K. Baruah, “Scheduling arbitrary deadline sporadic task systems on multiprocessors”. In proceedings Real-Time Systems Symposium pages 384-393, 2008.

[13] B. Andersson, K. Bletsas, "Sporadic Multiprocessor Scheduling with Few Preemptions,", Euromicro Conference on Real-Time Systems (ECRTS), pp. 243-252, 2008.

[14] B. Andersson, “Global static-priority preemptive multiprocessor scheduling with utilization bound 38%.” In proceedings of 12th International Conference on Principles of Distributed Systems, 2008.

[15] N.C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", *Technical Report YCS 164*, Dept. Computer Science, University of York, UK, December 1991.

[16] N.C. Audsley, A. Burns, R.I. Davis, K. W. Tindell, A. J. Wellings, “Fixed Priority scheduling an Historical perspective”, Real-Time Systems 8(3). pp. 173-198. 1995.

[17] N.C. Audsley, “On priority assignment in fixed priority scheduling”, Information Processing Letters, 79(1): 39-44, May 2001.

[18] T.P. Baker, “Stack-based Scheduling of Real-Time Processes.” Real-Time Systems Journal (3)1, pp. 67-100, 1991.

[19] T.P. Baker, “Multiprocessor EDF and deadline monotonic schedulability analysis”. In Proc. 24th IEEE Real-Time Systems Symposium, pp. 120–129, Cancun, Mexico, 2003.

[20] T.P. Baker, “An analysis of EDF scheduling on a multiprocessor”. IEEE Trans. on Parallel and Distributed Systems, 15(8):760–768, Aug. 2005.

[21] T.P. Baker, “An analysis of fixed-priority scheduling on a multiprocessor”. Real Time Systems, 32(1-2), 49-71, 2006.

[22] T.P. Baker, “A comparison of global and partitioned EDF schedulability tests for multiprocessors”. In Proceedings of the International Conference on Real-Time and Network Systems (RTSN). 119—130, 2006.

[23] T.P. Baker, M. Cirinei, “A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks”, Proceedings of the Work-In-Progress (WIP) session of the 27th IEEE Real-Time Systems Symposium (RTSS’06) (Rio de Janeiro, Brazil), December 2006.

[24] T.P. Baker, S.K. Baruah, “Schedulability analysis of multiprocessor sporadic task systems”, Handbook of Real-Time and Embedded Systems (2007), Edited by I. Lee, Joseph Y.-T. Leung and S. H. Son. Chapman Hall/CRC Press.

[25] T.P. Baker, S.K. Baruah, “Schedulability Analysis of Multiprocessor Sporadic Task Systems”. Technical report TR-060601. March 2007.

[26] T.P.Baker, M. Cirinei, M. Bertogna, “EDZL scheduling analysis”. Real-Time Systems. 40:3, 264-289, 2008

[27] T.P. Baker, S.K. Baruah, “Schedulability Analysis of global EDF”, Real-Time Systems, 38: 223-235, 2008.

[28] T.P. Baker, S.K. Baruah. “Sustainable multiprocessor scheduling of sporadic task systems”. In Proceedings of the EuroMicro Conference on Real-Time Systems, pp. 141-150, 2009.

[29] S.K. Baruah., A.K. Mok, L.E. Rosier, “Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor”. *In Proceedings of the IEEE Real-Time System Symposium*, pages182-190, 1990.

[30] S.K. Baruah, L.E. Rosier, R.R Howell, “Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on one Processor”. Real-Time Systems, 2(4), pages 301-324, 1990.

[31] S.K. Baruah, J. Gehrke, C. G. Plaxton, “Fast scheduling of periodic tasks on multiple resources”. In Proceedings of The International Parallel Processing Symposium, April. 1995.

[32] S.K. Baruah, N. Cohen, G. Plaxton, D. Varvel, “Proportionate progress: A notion of fairness in resource allocation”, Algorithmica 15, no. 6, pp. 600–625, 1996.

[33] S.K. Baruah, J. Goossens, “Rate-monotonic scheduling on uniform multiprocessors”. IEEE Transactions on Computers 52(7): 966–970 2003.

[34] S.K. Baruah, J. Carpenter, “Multiprocessor Fixed-Priority Scheduling with Restricted Interprocessor Migrations” In Proceedings. ECRTS pp. 195-202, 2003.

[35] S.K. Baruah, N. Fisher, “Partitioned multiprocessor scheduling of sporadic task systems”. In Proceedings of the. 26th Real-Time Systems Symposium, Dec. 2005.

[36] S.K. Baruah, N. Fisher, “The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems”. IEEE Transactions on Computers, 55(7):918–923, July 2006.

[37] S.K. Baruah., A. Burns, “Sustainable Scheduling Analysis”. In Proceedings of the IEEE Real-Time Systems Symposium, pp. 159-168, 2006.

[38] S.K. Baruah, “Techniques for Multiprocessor Global Schedulability Analysis”. In proceedings of the Real-Time Systems Symposium, pp. 119-128, 2007.

[39] S.K. Baruah, Fisher, N. W. “The partitioned dynamic-priority scheduling of sporadic task systems”. Real-Time Systems. 36, 3 (Aug. 2007), 199-226.

[40] S.K. Baruah, N. Fisher, “Non-migratory feasibility and migratory schedulability analysis of multiprocessor real-time systems”. Real-Time Systems. 39, 1-3 (Aug. 2008).

[41] S.K. Baruah, N. Fisher. “Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems.” Proceedings of the 9th International Conference on Distributed Computing and Networking, Kolkata, India. January 2008.

[42] S.K. Baruah, J. Goossens. “The EDF scheduling of sporadic task systems on uniform multiprocessors”. In proceedings of the Real-Time Systems Symposium, pp. 367-374, 2008.

[43] S.K.Baruah, T.P. Baker, Global EDF schedulability analysis of arbitrary sporadic task systems. In Proceedings of the EuroMicro Conference on Real-Time Systems, pp. 3–12, 2008.

[44] S.K. Baruah, T.P. Baker, “An analysis of global EDF schedulability for arbitrary sporadic task systems. Real-Time Systems ECRTS special issue, to appear 2009.

[45] S.K. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, “Implementation of a speedup-optimal global EDF schedulability test”. In Proceedings of the EuroMicro Conference on Real-Time Systems, pp.259-268, 2009.

[46] S.K. Baruah, “Schedulability analysis of global deadline monotonic scheduling”. Technical report available from http://www.cs.unc.edu/~baruah/Pubs.shtml.

[47] M. Bertogna, M. Cirinei, G. Lipari, “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, Pisa, Italy, Dec. 2005.

[48] M. Bertogna, M. Cirinei, G. Lipari, “Improved schedulability analysis of EDF on multiprocessor platforms”. In Proceedings of the 17th Euromicro Conference on Real-Time Systems, pp. 209–218, 2005.

[49] M. Bertogna, “Real-Time Scheduling for Multiprocessor Platforms”. PhD Thesis, Scuola Superiore Sant’Anna, Pisa, 2007.

[50] M. Bertogna, M. Cirinei, “Response Time Analysis for global scheduled symmetric multiprocessor platforms”. In proceedings of the Real-Time Systems Symposium, pp. 149-158, 2007.

[51] M. Bertogna, M. Cirinei, G. Lipari, “Schedulability analysis of global scheduling algorithms on multiprocessor platforms”. IEEE Transactions on parallel and distributed system, June 2008.

[52] M. Bertogna, “Evaluation of existing schedulability tests for global EDF”. First International Workshop on Real-time Systems on Multicore Platforms: Theory and Practice 2009

[53] V. Bharadwaj, T. G. Robertazzi, D. Ghose. “Scheduling Divisible Loads in Parallel and Distributed Systems”. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[54] E. Bini, G.C. Buttazzo. “Measuring the Performance of Schedulability tests”. Real-Time Systems, 30(1–2):129–154, May 2005.

[55] E. Bini, G.C. Buttazzo, M. Bertogna, “The Multy Supply Function Abstraction for Multiprocessors”. In proceedings of the Real-Time Computing Systems and Applications, 2009.

[56] E. Bini, M. Bertogna, S.K. Baruah, “Virtual Multiprocessor Platforms: Specification and Use”. In proceedings of the Real-Time Systems Symposium, *to appear* 2009.

[57] K. Bletsas, B. Andersson, “Notional Processors: An Approach for Multiprocessor Scheduling”. In proceedings of the Real-Time Applications Symposium, pp. 3-12, 2009.

[58] A. Block, H. Leontyev, B. Brandenburg, J.H. Anderson, “A Flexible Real-Time Locking Protocol for Multiprocessors”. In Proceedings of RTCSA, pp 47-56. 2007.

[59] B.B. Brandenburg, J.H. Anderson, “A Comparison of the M-PCP, D-PCP, and FMLP on LITMUS” In proceedings of the International Conference On Principles Of Distributed Systems 2008.

[60] B. Brandenburg, J. Calandrino, J. Anderson, “On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study”. In proceedings of the Real-Time Systems Symposium, pp. 157-169, 2008.

[61] B.B. Brandenburg, J.M. Calandrino, A. Block, H. Leontyev, J. Anderson, “Real-Time Sychronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin”. In Proceedings of the Real-Time and Embedded Technology and Applications Symposium, pp. 342-353, 2008.

[62] A. Burchard, J. Liebeherr, Y. Oh, S.H. Son, “New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems”, IEEE Transactions on Computers, vol. 44, number 12, Dec. 1995.

[63] A. Burns, A. Wellings, “Real-Time Systems and Programming Languages (Fourth Edition) Ada 2005, Real-Time Java and C/Real-Time POSIX” Addison Wesley Longmain, ISBN: 0201729881, 2009.

[64] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, “A constant-approximate feasibility test for multiprocessor real-time scheduling”. In Proceedings of the European Symposium on Algorithms, pp. 210–221, 2008.

[65] G. Buttazzo, “Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications”, 2nd Edition, Springer, ISBN: 978-0-387-23137-2, 2005.

[66] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, S.K. Baruah. “A categorization of real-time multiprocessor scheduling problems and algorithms” In Handbook of Scheduling: Algorithms, Models, and Performance Analysis, 2004.

[67] Y-H Chao, S-S Lin, K-J Lin, “Schedulability issues for EDZL scheduling on real-time multiprocessor systems”, Information Processing Letters, Volume 107, Issue 5, pp. 158-164, 16 August 2008

[68] C-M. Chen, S.K. Tripathi, A. Blackmore, “A Resource Synchronization Protocol for Multiprocessor Real-Time Systems” In Proceedings of the International Conference on Parallel Processing, pp. 159-162, 1994.

[69] H. Cho, B. Ravindran, E.D. Jensen, “An Optimal Real-Time Scheduling Algorithm for Multiprocessors”. In Proceedings of the Real-Time Systems Symposium pp. 1001-110, 2006.

[70] M. Cirinei, T. P. Baker. “EDZL scheduling analysis”. In proceedings of the EuroMicro Conference on Real-Time Systems, pp. 9–18 , 2007.

[71] S. Collette, L. Cucu, J. Goossens, “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, pp. 123-128, 2007.

[72] S. Collette, L. Cucu and J. Goossens, ”Integrating Job Parallelism in Real-Time Scheduling Theory”. Information Processing Letters, vol. 106(5): 180-187, May 2008.

[73] L. Cucu, J. Goossens, "Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors", In proceedings of the International Conference on Emerging Technologies and Factory Automation, 2006

[74] L. Cucu, J. Goossens, "Feasibility Intervals for Multiprocessor Fixed-Priority Scheduling of Arbitrary Deadline Periodic Systems ", In proceedings of the 10th Design, Automation and Test in Europe, 2007

[75] L. Cucu, "Optimal priority assignment for periodic tasks on unrelated processors", In proceedings of the Euromicro Conference on Real-Time Systems, WIP session, 2008.

[76] S. Davari, S.K. Dhall, “On a Periodic Real Time Task Allocation Problem”, Annual International Conference on System Sciences, 1986.

[77] R.I. Davis, A. Burns, “Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems” In proceedings of Real-Time Systems Symposium, *to appear*, 2009.

[78] S. K. Dhall, C. L. Liu, “On a Real-Time Scheduling Problem”, Operations Research, vol. 26, number 1, pp. 127-140, 1978.

[79] M.L. Dertouzos, “Control Robotics: The Procedural Control of Physical Processes”. *In Proceedings of the IFIP congress*, pages 807-813, 1974.

[80] M.L. Dertouzos, A.K. Mok, “Multiprocessor scheduling in a hard real-time environment”. IEEE Transactions on Software Engineering, 15(12):1497–1506, 1989.

[81] U.C. Devi, H. Leontyev, J. Anderson, “Efficient synchronization under global EDF scheduling on multiprocessors”. In proceedings Euromicro conference on Real-Time Systems, pp. 75-84, 2006.

[82] J. Edmonds, K. Pruhs, “Scalably scheduling processes with arbitrary speedup curves”. In Proceedings of the Symposium on Discrete Algorithms, pp.685-692, 2009.

[83] N. Fisher, S. Baruah, T. Baker. “The partitioned scheduling of sporadic tasks according to static priorities”. In Proceedings of the EuroMicro Conference on Real-Time Systems, pp. 118–127, 2006.

[84] N. Fisher, S.K. Baruah. “Global Static-Priority Scheduling of Sporadic Task Systems on Multiprocessor Platforms.” In Proceedings of the International Conference on Parallel and Distributed Computing and Systems, 2006.

[85] N. Fisher, S.K. Baruah, “The Global Feasibility and Schedulability of General Task Models on Multiprocessor Platforms”. In Proceedings of the Euromicro Conference on Real-Time Systems, pp51-60, 2007.

[86] N. Fisher, The multiprocessor real-time scheduling of general task systems, Ph.D. thesis, Department of Computer Science, The University of North Carolina at Chapel Hill, 2007.

[87] K. Funaoka, S. Kato, N. Yamasaki, “Work-Conserving Optimal Real-Time Scheduling on Multiprocessors” In proceedings of the Euromicro Conference on Real-Time Systems pp. 13-22, July 2008.

[88] S. Funk, J. Goossens, S.K. Baruah. On-line scheduling on uniform multiprocessors. In IEEE, editor, Proceedings of the IEEE Real-Time Systems Symposium, pages 183–192, Dec 2001.

[89] S. Funk, V. Nadadur, “LRE-TL: An Optimal Multiprocessor Algorithm for Sporadic Task Sets”. In proceedings Real-Time Networks and Systems conference, pp. 159-168, 2009.

[90] P. Gai, G. Lipari, M. Di Natale, “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, pp. 73-83, 2001.

[91] P. Gai, M.D. Natale, G. Lipari, A. Ferrari, C. Gabellini, P. Marceca, “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 2003.

[92] M. Garey, D. Johnson, “Computers and Intractability: a Guide to the Theory of NP-Completeness”. W. H. Freeman and company, NY,1979.

[93] R. L. Graham, “Bounds on multiprocessor scheduling anomalies and related packing problem”. In Proceedings AFIPS Spring Joint Computer Conference, pp. 205–217, 1972.

[94] J. Goossens, S. Funk, S.K. Baruah. “Priority-driven scheduling of periodic task systems on multiprocessors”. Real Time Systems, 25(2–3):187–205, Sept. 2003.

[95] N. Guan, M. Stigge, W.Yi, G. Yu, “New Response Time Bounds for Fixed Priority Multiprocessor Scheduling”. In

proceedings of the Real-Time Systems Symposium, to appear 2009.

[96] P. Holman, J. H. Anderson, “Adapting Pfair scheduling for symmetric multiprocessors”. Journal of Embedded Computing 1(4): 543-564 (2005).

[97] K. Hong, J. Leung, “On-line scheduling of real-time tasks”. In Proceedings of the Real-Time Systems Symposium, pp. 244–250, 1988.

[98] K.S. Hong, J.Y.-T Leung, “On-line scheduling of real-time tasks”, IEEE Transactions on Computers 41, pp. 1326–1331, 1992

[99] W.A. Horn, “Some simple scheduling algorithms”, Naval Research Logistics Quarterly 21 pp.177–185, 1974.

[100] R. Ha, J. W.-S. Liu, “Validating timing constraints in multiprocessor and distributed real-time systems”. In Proceedings of the International conference on Distributed Computing Systems, pp. 162–171, 1994.

[101] B. Kalyanasundaram, K. Pruhs, “Speed is as powerful as clairvoyance”. In Proceedings of the Symposium on Foundations of Computer Science, pp. 214-221, 1995.

[102] S. Kato, N. Yamasaki. “Real-Time Scheduling with Task Splitting on Multiprocessors”. In Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 441-450, 2007.

[103] S. Kato, N. Yamasaki, “Portioned EDF-based Scheduling on Multiprocessors”, EMSOFT pp. 139-148, 2008.

[104] S. Kato, N. Yamasaki. “Portioned Static-Priority Scheduling on Multiprocessors”. In Proceedings of the International Parallel and Distributed Processing Symposium, 2008.

[105] S. Kato, N. Yamasaki. “Semi-Partitioned Fixed-Priority Scheduling on Multiprocessors”. In Proceedings of the Real-Time and Embedded Technology and Applications Symposium, pp. 23-32, 2009.

[106] S. Kato, N. Yamasaki, Y. Ishikawa. “Semi-Partitioned Scheduling of Sporadic Task Systems on Multiprocessors”. In Proceedings of the Euromicro Conference on Real-Time Systems pp. 249-258, 2009.

[107] A. Khemka, R. K. Shyamasundar, “An Optimal Multiprocessor Real-Time Scheduling Algorithm”. Journal of Parallel and Distributed Computing 43(1): 37-45 (1997).

[108] K. Lakshmanan, R. Rajkumar, J. Lehoczky, "Partitioned Fixed-Priority Preemptive Scheduling for Multi-core Processors", In proceedings of the Euromicro Conference on Real-Time Systems, pp.239-248, 2009.

[109] D. Lammers, “Intel cancels Tejas, moves to dual-core designs” EETimes, May 7th 2004.

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

[111] S.K. Lee, “On-line multiprocessor scheduling algorithms for real-time tasks”, in IEEE Region 10’s Ninth Annual International Conference, pp. 607–611, 1994,

[112] W. Y. Lee, S. J. Hong, J. Kim, “On-line scheduling of scalable real-time tasks on multiprocessor systems”. Journal of Parallel and Distributed Computing, 63(12):1315–1324, 2003.

[113] J. Lehoczky. “Fixed priority scheduling of periodic task sets with arbitrary deadlines”. In Proceedings of the Real-Time Systems Symposium, pp. 201–209, 1990.

[114] H. Leontyev, J.H. Anderson, “Hierarchical Multiprocessor Bandwidth Reservation Scheme with Timing Guarantees.” In proceedings of the Euromicro Conference on Real-Time Systems pp. 191-200, 2008.

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

[116] C.L. Liu, “Scheduling algorithms for multiprocessors in a hard real-time environment”. JPL Space Programs Summary, vol. 37-60, pp. 28-31, 1969.

[117] C.L. Liu, J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment ", Journal of the ACM, 20(1): 46-61, Jan. 1973.

[118] J.W.S. Liu, “Real Time Systems” Prentice Hall, ISBN 978-0130996510, 2000.

[119] J. Y.-T. Leung, J. Whitehead, “On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks”, Performance Evaluation, number 2, pp. 237-250, 1982.

[120] J. M. Lopez, M. Garcia, J. Diaz, D. Garcia, “Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems”. In Proceedings of the Euromicro Conference on Real-time Systems, pp. 25–33, 2000.

[121] J. M. Lopez, J. L. Diaz, M. Garcia, D. F. Garcia, “Utilization bounds for multiprocessor RM scheduling”. Real-Time Systems, 24(1):5–28, 2003.

[122] J. M. Lopez, J.L. Dıaz, D.F. Garcıa, “Minimum and Maximum Utilization Bounds for Multiprocessor Rate Monotonic Scheduling”, IEEE Transactions on Parallel and Distributed Systems, 15(7), July 2004.

[123] J.M. Lopez, M. Garcıa, J.L. Dıaz, D.F. Garcıa, “Utilization Bounds for EDF scheduling on Real-Time Multiprocessor Systems”, Real Time Systems, vol. 28, Issue 1, Oct. 2004.

[124] L. Lundberg, “Analyzing Fixed-Priority Global Multiprocessor Scheduling”. In proceedings of the Real-Time and Embedded Technology and Applications Symposium, 2002.

[125] J. Mellor-Crummey, M. Scott. “Algorithms for scalable synchronization on shared-memory multiprocessors”. ACM Transactions on Computer Systems, 9(1):21–65, February 1991.

[126] Y. Oh, S.H. Son, “Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem”, Technical Report CS-93-24. Univ. Of Virginia. Dept. of Computer Science, May 1993.

[127] Y. Oh, S. H. Son, “Fixed Priority Scheduling of Periodic Tasks on Multiprocessor Systems”, Tech. Report CS-95-16, Univ. Of Virginia. Dept. of Computer Science, March 1995.

[128] D. I. Oh, T. P. Baker. “Utilization bounds for N-processor rate monotone scheduling with stable processor assignment”. Real Time Systems, 15(2):183–193, Sept. 1998.

[129] C.A. Phillips, C. Stein, E. Torng, J. Wein, “Optimal time-critical scheduling via resource augmentation”. In Proceedings of the ACM Symposium on theory of Computing 1997.

[130] R. Rajkumar, L. Sha, J. P. Lehoczky, “Real-time synchronization protocols for multiprocessors”. In proceedings of the Real Time Systems Symposium, pp. 259-269, 1988.

[131] T. G. Robertazzi, “Ten reasons to use divisible load theory” Computer,36(5):63–68, 2003.

[132] T. Rothvoss, “On the computational complexity of periodic scheduling”. Ph.D. Thesis, Ecole Polytechnique Federale de Lausanne, 2009.

[133] S. Saez, J. Vila, A. Crespo. “Using exact feasibility tests for allocating real-time tasks in multiprocessor systems”. In Proceedings of the Euromicro Workshop on Real-time Systems, pp. 53– 60, 1998.

[134] L. Sha, R. Rajkumar, J.P. Lehoczky. “Priority inheritance protocols: An approach to real-time synchronization”. IEEE Transactions on Computers, 39(9): 1175-1185, 1990.

[135] L. Sha, T. Abdelzaher, K-E. Arzen, A. Cervin, T.P. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A.K. Mok, “Real Time Scheduling Theory: A Historical Perspective”, Real-Time Systems, Vol 28, No 2/3, pp101-155, 2004.

[136] I. Shin, A. Easwaran, I. Lee, “Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors”, In proceedings of the Euromicro Conference on Real-Time Systems pp. 181-190, 2008.

[137] H.R. Simpson, “Four-Slot Fully Asynchronous Communication Mechanism”. IEE Proceedings, 137 Part E(1):17–30, Jan. 1990.

[138] A. Srinivasan, S.K. Baruah, “Deadline-based scheduling of periodic task systems on multiprocessors”. Information Processing Letters, 84:93–98, 2002.

[139] B. Veeravalli, D. Ghose, T. G. Robertazzi, “Divisible load theory: A new paradigm for load scheduling in distributed systems”. Cluster Computing, 6(1):7–17, 2003.

[140] D. Zhu, D. Mossé, R.G. Melhem, “Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary?” In proceedings of the Real Time Systems Symposium, pp. 142-151, 2003.