Sistemas de Tempo Real:

Multiprocessadores: Visão Geral

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo

References

- A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems
  - Robert I. Davis and Alan Burns
  - University of York, Department of Computer Science, relatório técnico, 2009

Motivation 1/5

- Companies building embedded real-time systems aim to meet the needs and desires of their customers
- Must provide systems that are
  - more capable, more flexible, and more effective than their competition
  - and to bring these systems to market earlier
- This desire for technological progress has resulted in a rapid increase in
  - software complexity
  - the processing demands placed on the underlying hardware

Motivation 2/5

- In the past silicon vendors addressed demands for increasing processor performance through miniaturisation to increase processor clock speeds
- This approach has led to problems with both
  - high power consumption
  - excessive heat dissipation
- 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:
  - 7th May 2004
  - Intel cancelled the successor to the Pentium P4 processor called Tejas
  - Because extremely high power consumption

Motivation 3/5

- Dynamic power consumption is a dominant factor for chip designs using technology above the 100nm level
  - the power lost charging and discharging capacitive load)
- Transistor leakage current becomes important for sub 100nm technology
  - 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 miniaturization
- This problem can be partially ameliorated by running at a lower voltage
- Reducing voltage also limits the maximum operating frequency, restricting performance
- A solution to this problem is to limit minaturization and operating frequencies and to use multiple processors on a single chip

Motivation 4/5

- On 27th July 2006, two years after cancellation of Tejas, Intel officially released the Core Duo processor
- In future, it is expected that highend 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
  - It has 80 processor cores providing 1 Teraflop performance at 3 GHz

Motivation 5/5

- List of multicore processors (2009):
  - 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.

Introduction

- In 1969, C. L. Liu 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."

Classification of multiprocessor systems

- Multiprocessor systems can be classified into three categories:

- Heterogeneous
  - The processors are different
  - The rate of execution of a task depends on both the processor and the task
  - Not all tasks may be able to execute on all processors
- Homogeneous (this paper)
  - The processors are identical
  - The rate of execution of all tasks is the same on all processors
- Uniform
  - The rate of execution of a task depends only on the speed of the processor
  - A processor of speed 2 will execute all tasks at exactly twice the rate of a processor of speed 1

Task Model

- 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)

- Most research work into multiprocessor real-time scheduling focuses on two 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)

Periodic Task Model

- Periodic task model
  - The jobs of a task arrive strictly periodically, separated by a fixed time interval

- Periodic task sets may be classified as:
- Synchronous
  - If there is some point in time at which all of the tasks arrive simultaneously

- Asynchronous
  - Task arrival times are separated by fixed offsets and there is no simultaneous arrival time

Sporadic Task Model

- Sporadic task model
  - Each job of a task may arrive at any time once a minimum inter-arrival time has elapsed

- In the sporadic task model:
  The arrival times of the jobs of different tasks are assumed to be independent

Task Model 1/5

- Intra-task parallelism is not permitted by either model
- At any given time, each job may execute on at most one processor
- It is assumed that only a single job of a task is ready to execute at any given time
- It is assumed that once a job starts to execute it will not suspend itself

Task Model 2/5

- Each task τ_i is characterized by:
  - Relative *deadline* D_i
  - *Worst-case execution time* C_i
  - Minimum interarrival time or *period* T_i
- The *utilization* u_i , of task τi is given by C_i / T_i
- The utilisation u_{sum} of a taskset is the sum of the utilizations of all of its tasks
- A task's *worst-case response time* R_i , is defined as the longest time from a job of that task arriving to it completing execution
- The *hyperperiod* $H(\tau)$ of a taskset is defined as the least common multiple of the task periods

Task Model 3/5

- There are three levels of constraint on task deadlines that are studied in the literature
- *Implicit deadlines*:
  - all task deadlines are equal to their periods ($D_i = T_i$)
- *Constrained deadlines*:
  - all task deadlines are less than or equal to their periods ($D_i \leq T_i$)
- *Arbitrary deadlines*:
  - task deadlines may be less than, equal to, or greater than their periods

Task Model 4/5

- 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
- Some policies 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

Task Model 5/5

- In the case of global scheduling:
- A job may migrate from one processor to another
  - As a result of pre-emption and subsequent resumption
- The cost of preemption, 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

Taxonomy of Multiprocessor Scheduling Algorithms 1/6

- Multiprocessor scheduling can be viewed as attempting to solve two problems:
- The *allocation problem*
  - on which processor a task should execute
- The *priority problem*
  - when, and in what order with respect to jobs of other tasks, should each job execute

Taxonomy of Multiprocessor Scheduling Algorithms 2/6

- About Allocation:
- *No migration*
  - Each task is allocated to a processor and no migration is permitted
- *Task-level migration*
  - The jobs of a task may execute on different processors
  - However each job can only execute on a single processor
- *Job-level migration*
  - A single job can migrate to and execute on different processors
  - However parallel execution of a job is not permitted

Taxonomy of Multiprocessor Scheduling Algorithms 3/6

- About Priority:
- *Fixed task priority*
  - Each task has a single fixed priority applied to all of its jobs
- *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
- *Dynamic priority*
  - A single job may have different priorities at different times
  - For example Least Laxity First (LLF) scheduling

Taxonomy of Multiprocessor Scheduling Algorithms 4/6

- Scheduling algorithms where no migration is permitted are referred to as *partitioned*
- Those where migration is permitted are referred to as *global*
- The majority of research into global scheduling algorithms has focussed on models where arbitrary migration (job-level migration) is permitted
- In this paper the term *global* is used to mean job-level migration
  - clarification is provided when only task-level migration is permitted

Taxonomy of Multiprocessor Scheduling Algorithms 5/6

- A scheduling algorithm is said to be *workconserving*
  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 workconserving
  - a processor may become idle
  - but cannot be used by ready tasks allocated to a different processor

Taxonomy of Multiprocessor Scheduling Algorithms 6/6

- Scheduling algorithms can be further classified as:
- *Pre-emptive*
  - Tasks can be pre-empted by a higher priority task at any time
- *Non-pre-emptive*
  - Once a task starts executing, it will not be pre-empted
  - It will execute until completion
- *Co-operative*
  - Tasks may only be pre-empted at defined scheduling points within their execution
  - Execution of a task consists of a series of non-pre-emptable sections
- This survey focus on pre-emptive scheduling algorithms

Schedulability, Feasibility, and Optimality 1/3

- 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

Schedulability, Feasibility, and Optimality 2/3

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

Schedulability, Feasibility, and Optimality 3/3

- 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
- 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*

Processor Demand Function

- 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

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

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

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

- 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

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

Performance Metrics

- Four performance metrics have been used to compare the effectiveness of different multiprocessor scheduling algorithms / schedulability analyzes
- Utilisation bounds
- Approximation Ratio
- Resource Augmentation or Speedup factor
- Empirical measures, such as the percentage of tasksets that are found to be schedulable

Utilization bounds

- Worst-case *utilization bounds* are a useful performance metric for implicit-deadline tasksets
- 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
- There exist implicit-deadline tasksets with total utilization infinitesimally greater than U_A that are unschedulable according to algorithm A
- There are no implicit-deadline tasksets with total utilisation $u_{sum} \le U_A$ that are unschedulable according to algorithm A
- U_A can be used as a simple sufficient (but not necessary) schedulability test

Approximation Ratio

- The *approximation ratio* is a way of comparing the performance of a scheduling algorithm A with that of an optimal algorithm
- The number of processors required according to an optimal algorithm is $M_O(\tau)$
- The number required according to algorithm A *is* $M_A(\tau)$
- The approximation ratio $\Re_A$ of algorithm A is: $\Re_A = \lim_{M_O \to \infty}\left(\max_{\forall \tau}\left(\frac{M_A}{M_O}\right)\right)$
- Note that $\Re_A \ge 1$
  - Smaller values indicates a more effective scheduling algorithm
  - $\Re_A = 1$ implies an optimal algorithm
- Scheduling algorithms are referred to as *approximate*
  - if they have a finite approximation ratio

Speedup Factor 1/2

- The *resource augmentation* factor *f* or *speedup factor f* is an alternative method of comparing the performance of a scheduling algorithm *A* with an optimal algorithm
- The resource augmentation factor considers the increase in processing speed that would be required to obtain schedulability under algorithm A
- The resource augmentation 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 on *m* processors of speed 1
  - become schedulable under algorithm *A*

Speedup Factor 2/2

- 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 *speedup factor* f_A for algorithm *A* is: $f_A = \max_{\forall m, \forall \tau}(f(\tau))$
  - $f_A \geq 1$
  - smaller values indicate a more effective algorithm
  - $f_A = 1$ implies an optimal algorithm

Empirical Measures 1/2

- A comparative measure of the effectiveness
  of different scheduling algorithms and their analyzes
  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,
  - Exact feasibility tests are not known for the case of sporadic tasksets
  - They are potentially intractable for periodic tasksets
- It compares the relative performance of two or more sufficient schedulability tests/scheduling algorithms
- It is important to use a taskset generation algorithm that is unbiased

Empirical Measures 2/2

- Simulation cannot, in general, prove schedulability
- It can prove that a taskset is unschedulable if the simulation reveals a deadline miss
- Simulation can be used as a sufficient test of un-schedulability

Fundamental Results

- **Fundamental results**
  - Results about multiprocessor real-time scheduling that are independent of specific scheduling algorithms
- Optimality
- Feasibility
- Comparability
- Predictability
- Sustainability
- Anomalies

Optimality 1/3

- 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
  - 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
  - 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

Optimality 2/3

- In 1988, Hong and Leung 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
  - optimality is impossible without clairvoyance
- In 1989, this result was extended by Dertouzos and Mok who showed that knowledge of arrival times is necessary for optimality
  - even if execution times are known

Optimality 3/3

- In 2007, Fisher 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

Feasibility 1/7

- In 1974, Horn observed that $u_{sum} \leq m$ 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 is: $load(\tau) \leq m$

Feasibility 2/7

- In 2006, Baker and Cirinei improved upon this necessary feasibility condition by considering the modified processor load
  - 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

    $load^*(\tau) \leq m$
- They showed that an upper bound on the modified processor load $load^*(\tau)$ can be found by considering a synchronous arrival sequence
  - $load^*(\tau)$ calculated from the modified processor demand bound function for each task

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

Feasibility 3/7

- In 2006, Cucu and Goossens 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* algorithm
  - *memoryless* = makes decisions based only on the currently ready tasks (EDF)
- An exact schedulability test can be obtained by checking if the schedule generated misses any deadlines in (0,H]
- An exact feasibility test for fixed job priority (EDF) scheduling could 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
- No exact feasibility test for sporadic tasksets under a fixed-job priority algorithm

Feasibility 4/7

- In 2007, Cucu and Goossens investigated the feasibility for fixed-task priority algorithms
- The taskset hyperperiod (0,H] is a feasibility interval
  - For implicit- and constrained-deadline synchronous periodic tasksets
  - Fixed-task priority algorithms are both deterministic and memoryless
- For arbitrary deadline periodic tasksets
  the hyperperiod (0,H] is a feasibility interval provided that
  - all previously released jobs are completed by H
- For asynchronous, periodic task systems,
  - Longer intervals are required to prove exact schedulability

Feasibility 5/7

- 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 scheduled using fixed task priorities
- However, this approach quickly becomes intractable as taskset cardinality increases
- No exact feasibility test or optimal priority ordering algorithm is known for sporadic tasksets scheduled using fixed task priorities

Feasibility 6/7

- In 2007, Fisher and Baruah 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
  - It proves that such an algorithm exists but not what the algorithm is
- The test for sporadic tasksets with arbitrary deadlines, is sufficient
  - there are tasksets which it deems infeasible which are in fact feasible

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

- The speedup factor is $1/(\sqrt{2}-1) \approx 2.41$
  - any sporadic taskset that is feasible on m processors of speed $(\sqrt{2}-1)$ will be feasible by the test on m processors of unit speed

Feasibility 7/7

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

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

Comparability 1/3

- Comparing the tasksets that can be scheduled by two different multiprocessor scheduling algorithms *A* and *B*
- *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*
  - tasksets exist that are schedulable according to *A*, but not according to *B*
- *Equivalence*:
  - if all of the tasksets that are schedulable according to algorithm *B* are also schedulable according to algorithm *A*, and viceversa
- *Incomparable*:
  - Tasksets exist that are schedulable according to algorithm *A,* but not according to algorithm *B* and vice-versa

Comparability 2/3

- In 2004, Carpenter et al. considered the relationships between different classes of multiprocessor scheduling algorithm
- 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*
- All three partitioned classes (fixed task priority, fixed job priority, and dynamic priority) are incomparable with respect to all three task-level migration classes

Comparability 3/3

- 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 utilization bounds
  - applicable to periodic tasksets with implicit-deadlines
-

Class	Maximum utilization bound
Global (job-level migration) dynamic priority	m
All other classes	(m +1) / 2

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
- In real systems task execution times are almost always variable up to some worst-case value
- All priority driven pre-emptive scheduling algorithms for multiprocessor systems are predictable
  - fixed task priority or fixed job priority
- For any dynamic priority scheduling algorithm:
  it is necessary to consider/prove predictability before the algorithm can be considered useful

Sustainability 1/2

- 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
- 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 *selfsustainable* if such a modified taskset will always be deemed schedulable by the test

Sustainability 2/2

- 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
- 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

Anomalies

- A scheduling *anomaly* occurs when a change in taskset parameters results in a counter-intuitive effect on schedulability
- For example:
  - Increasing task periods, while keeping all other parameters constant, results in lower overall processor utilization
  - It is reasonable to expect it to improve schedulability
  - However, in some cases, this can result in the taskset becoming unschedulable
- This effect is referred to as a *period anomaly*
  - It is evidence of un-sustainability

Anomalies: Period and Execution Time 1/2

- 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 <u>increase in the period</u>
  or a <u>decrease in the worst-case execution time</u> of a task
  results in a <u>different allocation</u>
  which is then deemed to be <u>unschedulable</u>
- Such anomalies are known to exist for EDF scheduling
  - Example: FF (First Fit) and FFDU (First Fit Decreasing Utilization) allocation
- These anomalies also exist for many fixed task priority partitioning algorithms

Anomalies: Period and Execution Time 2/2

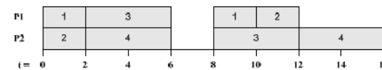
- Global fixed task priority scheduling of periodic task sets using an exact schedulability test is also subject to period anomalies
- The schedulability test is unsustainable with respect to increasing periods
- Period anomalies are known to exist for:
  - Global fixed task priority scheduling of synchronous periodic task sets
  - Global optimal scheduling (full migration, dynamic priorities) of synchronous periodic task sets

Anomalies: Critical Instant Effect 1/4

- 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
- In multiprocessor scheduling:
  the response time of a low priority task is longer when other higher priority tasks execute so that all processors are occupied
- The critical instant effect is a fundamental difference between global multiprocessor scheduling and partitioned / uniprocessor scheduling

Anomalies: Critical Instant Effect 2/4

- The task parameters (C_i , D_i ,T_i) are as follows:
  - τ 1 (2,2,8)
  - τ 2 (2,2,10)
  - τ 3 (4,6,8)
  - τ 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



Anomalies: Critical Instant Effect 3/4

- The exact response time of a task is dependent on both
  - the set of higher priority tasks *and*
  - their specific priority order
- A greedy approach to priority assignment
  - as used by Audsley's optimal priority assignment algorithm for the uniprocessor
- is not applicable to the multiprocessor case
  - when schedulability analysis uses exact response times
- It is applicable in conjunction with some sufficient schedulability tests

Anomalies: Critical Instant Effect 4/4

- The critical instant effect is also an issue in the analysis of global fixed job priority scheduling
- 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

Partitioned Scheduling 1/3

- Partitioned scheduling has the following advantages:
- 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
- Partitioned approaches use a separate run-queue per processor
  - For large systems, the overheads of manipulating a single global queue can become excessive

Partitioned Scheduling 2/3

- 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
- 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 task sets with implicit deadlines
  - Deadline Monotonic (DM) priority assignment is optimal for such task sets with constrained-deadlines
  - DM is not optimal for task sets with arbitrary deadlines or for asynchronous periodic task sets, however Audsley's priority assignment algorithm is optimal in these cases
- Considering pre-emptive uniprocessor scheduling using fixed job priorities:
  - EDF is the optimal scheduling algorithm for sporadic task sets independent of the deadline constraints

Partitioned Scheduling 3/3

- The main disadvantage of the partitioning approach to multiprocessor:
- The task allocation problem is analogous to the bin packing problem and is known to be NP-Hard

Partitioned Scheduling: Implicit-Deadline 1/10

- Early research into partitioned multiprocessor scheduling examined
  - EDF
  - RM
- Combined with bin packing heuristics for task allocation such as
  - “First-Fit” (FF)
  - “Next-Fit” (NF)
  - “Best-Fit” (BF)
  - “Worst-Fit” (WF)
  - “Decreasing Utilization” (DU)
  - “RMMatching” (2009), an $O(n3)$ algorithm
- Notation:
  RMBF means Rate Monotonic scheduling with Best Fit task allocation

Partitioned Scheduling: Implicit-Deadline 2/10

- **Approximation ratio** for periodic task sets with implicit-deadlines

Algorithm	Approximation Ratio ($\Re_A$)	
RMNF	2.67	
RMFF	2.33	
RMBF	2.33	
RM-FFDU	1.6666…	
FFDUF	2	
RMST	$1/(1- u_{max})$	u_{max}=highest task utilization
RMGT	1.75	
RMMatching	1.5	
EDF-FF	1.7	
EDF-BF	1.7	

Partitioned Scheduling: Implicit-Deadline 3/10

- **Approximation ratios** enable a comparison to be made between the different algorithms
- Their practical use as a schedulability test is severely limited
- Determining the minimum number of processors required by an optimal algorithm is an NP-hard problem
- The approximation ratio only holds as the number of processors required in the optimal case tends to infinity
- The utilization bounds that can be derived from these approximation ratios are pessimistic

Partitioned Scheduling: Implicit-Deadline 4/10

- For periodic task sets with implicit-deadlines, the largest worst-case **utilization bound** for any partitioning algorithm is:
  $U_{OPT} = (m + 1) / 2$
- This equation holds because m +1 tasks with execution time 1+ε and a period of 2 cannot be scheduled on m processors regardless of the allocation algorithm used
- The difficulties that partitioned scheduling has allocating large utilization tasks lead to significant research providing utilization bounds as a function of u_{max}
  - the highest utilisation of any task in the taskset

Partitioned Scheduling: Implicit-Deadline 5/10

- Burchard et al provided utilization bounds for the RMST (“Small Tasks”) algorithm
  - Attempts to place tasks with periods that are close to harmonics of each other on the same processor
- This algorithm favours tasks with utilization < 1/2:
  $U_{RMST} = (m - 2)(1- u_{max}) + 1 - \ln 2$
- Burchard et al also provided utilization bounds for the RMGT (“General Tasks”) algorithm
  - Separates tasks into two groups depending on whether their utilization is above or below 1/3

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

Partitioned Scheduling: Implicit-Deadline 6/10

- Oh and Baker showed that RM-FFDU has a utilization bound given by: $U_{RM-FFDU} = m(2^{1/2} - 1) \approx 0.41m$
- They also showed that the utilization bound for any fixed task priority partitioning algorithm is upper bounded by:

  $U_{OPT(FTP)} < (m+1) / (1 + 2^{1/(m+1)})$
- Lopez et al generalized 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

Partitioned Scheduling: Implicit-Deadline 7/10

- B. Andersson showed that the RBOUND-MP-NFR algorithm has a utilization bound of $U_{RBOUND-MP-NFR} = m/2$
- This result shows that a fixed task priority partitioning algorithm exists that is an optimal partitioning approach in the limited sense that its utilization bound is the maximum possible for any partitioning algorithm
- This does not mean that it is an optimal partitioning algorithm in the sense that it can schedule any task set that is schedulable according to any other partitioning algorithm

Partitioned Scheduling: Implicit-Deadline 8/10

- A reasonable allocation algorithm is one that only fails to allocate a task once there is no processor on which the task will fit
- Lopez et al showed that using EDF
- The lowest utilization bound for any *reasonable* allocation algorithm is given by:

  $L_{RA} = m - (m - 1)\, u_{max}$
- and that the highest utilization bound of any reasonable allocation algorithm is:

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

Partitioned Scheduling: Implicit-Deadline 9/10

- Lopez et al. showed that all reasonable allocation algorithms that order tasks by decreasing utilization achieve the higher limit
  - EDF-BF, EDF-FF
- EDF-WF, but not EDF-WFDU, achieves the lower limit
- When $u_{max} = 1$, $H_{RA} = \frac{(\lfloor 1/u_{max} \rfloor m + 1)}{(\lfloor 1/u_{max} \rfloor + 1)}$

  becomes $U_{OPT} = (m + 1) / 2$
- EDF-FF and EDF-BF are also 'optimal' partitioning approaches in the limited sense that their utilization bounds are as large as that of any partitioning algorithm

Partitioned Scheduling: Implicit-Deadline 10/10

- For applications with "small" tasks, then RMST and EDF-FF provide reasonably high utilization bounds
- For example, assuming $m = 10$ and $u_{max} = 0.25$
- The utilization bound for RMST is 63%
- The utilization bound for EDF-FF is 82%

Partitioned Scheduling: Constrained and Arbitrary Deadline 1/5

- Baruah and Fisher showed that EDF-FFD (decreasing density) is able to schedule any arbitrary-deadline sporadic task set provided that:

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

- The resource augmentation factor for EDF-FFD is *not* finite

Partitioned Scheduling: Constrained and Arbitrary Deadline 2/5

- Baruah and Fisher also developed an algorithm EDF-FFID based on ordering tasks by increasing relative deadline
- EDF-FFID is able to schedule any sporadic taskset with constrained deadlines provided that:

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

- For tasksets with arbitrary deadlines, the test becomes:

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

Partitioned Scheduling: Constrained and Arbitrary Deadline 3/5

- The resource augmentation or speedup factor required by EDF-FFID 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

Partitioned Scheduling: Constrained and Arbitrary Deadline 4/5

- Fisher et al. applied a similar approach to the problem of partitioning using fixed task priority scheduling using Deadline Monotonic
- The algorithm FFB-FFD (from the author's surnames), is based on
  - ordering tasks by decreasing relative deadline
  - 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 task set with constrained-deadlines provided th

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

- For tasksets with arbitrary deadlines, the test becomes:

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

Partitioned Scheduling: Constrained and Arbitrary Deadline 5/5

- The resource augmentation or speedup factor required by this algorithm is:
- $(3 - 1/m)$ for task sets with constrained deadlines
- $(4 - 2/m)$ for task sets with arbitrary deadlines

Global Scheduling 1/3

- Global scheduling has the following advantages
- There are typically fewer context switches / preemptions when global scheduling is used
  - The scheduler will only pre-empt a task when there are no processors idle
- Spare capacity created when a task executes for less than its worst-case execution time can be utilized by all other tasks
  - Not just those on the same processor.
- If a task overruns its WCET, there is a lower probability of deadline failure as worst-case behaviour of the entire system
  - Less probable than in a single processor
- Global scheduling is more appropriate for open systems
  - There is no need to run load balancing / task allocation algorithms when the set of tasks changes

Global Scheduling 2/3

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

Global Scheduling 3/3

- Dhall and Liu considered global scheduling of periodic task sets with implicit deadlines on m processors
- They showed that the utilization 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 utilization and one task with a longer period/deadline and utilization that approaches 1
- In 1997, Phillips et al. 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 task set schedulable by global EDF

Global Fixed Job Priority: Implicit Deadline 1/3

- B. Andersson et al. considered periodic task sets with implicit deadlines
- They showed that the maximum utilization bound for any global fixed job priority algorithm is:
  $U_{OPT} = (m + 1) / 2$
- Srinivasan and Baruah proposed the EDF-US[ς] algorithm that gives the highest priority to tasks with utilization greater than the threshold ς
- Setting the threshold to $m/(2m - 1)$ results in a utilization bound that is independent of $u_{\max}$:

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

Global Fixed Job Priority: Implicit Deadline 2/3

- Goossens et al. derived a utilization bound for global EDF applicable to periodic task sets with implicit-deadlines and showed that this bound is tight:
  $U_{EDF} = m - (m - 1)\, u_{\max}$
- Baruah and Carpenter showed that this same utilization bound applies to global EDF scheduling, assuming task level migration
- Goossens et al. also proposed an algorithm called EDF(k) that assigns the highest priority to the k tasks with the highest utilization
  - They showed a sufficient schedulability condition for EDF(k)
  - where u_k is the utilisation of the kth task, in order of decreasing utilisation

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

Global Fixed Job Priority: Implicit Deadline 3/3

- Baker showed that setting the threshold used in EDF-US[ς] to 1/2, results in the following utilization bound
  - which is the maximum possible bound for this class of algorithm

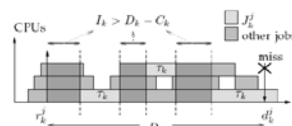
  $U_{EDF-US}[1/2] = (m + 1) / 2$
- Baker also proposed a variant of EDF(k) called EDF($k_{\min}$)
  - where $k_{\min}$ is the minimum value of k for which the sufficient test above holds
- Baker showed that the utilisation bound for EDF($k_{\min}$) is also
  $U_{EDF}[k_{\min}] = (m + 1) / 2$
- This is the maximum possible utilization bound for this class of scheduling algorithm
- However, EDF($k_{\min}$) dominates EDF-US[1/2] in terms of the task sets that it can schedule

Global Fixed Job Priority: Constrained and Arbitrary-Deadline 1/12

- The proof of the utilization bound was extended
  - by Bertogna et al. to the case of sporadic task sets with constrained deadlines and
  - by Baruah and Baker to the arbitrary-deadline case
  - giving the following sufficient schedulability test based on task density:
    $\delta_{sum} \leq m - (m - 1)\, \delta_{\max}$
- Bertogna also adapted the utilization separation approach of EDF-US to the case of sporadic task sets with constrained and arbitrary deadlines, 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 task set is schedulable according to EDF-DS[1/2] provided that:
  $\delta_{sum} \leq (m + 1) / 2$

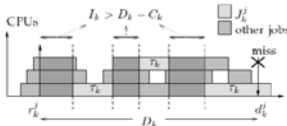
Global Fixed Job Priority: Constrained and Arbitrary-Deadline 2/12

- Baker developed a general strategy for determining the schedulability of sporadic task sets
- 1. Consider an interval, referred to as the *problem window*, at the end of which a deadline is missed
  - for example the interval $[r_k, d_k]$ from the arrival to the deadline of some job of task τ_k



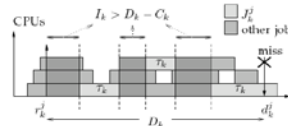
Global Fixed Job Priority: Constrained and Arbitrary-Deadline 3/12

- 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
  - *carry-in* jobs that have not completed execution before the start of the interval



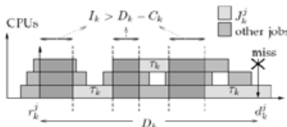
Global Fixed Job Priority: Constrained and Arbitrary-Deadline 4/12

- 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



Global Fixed Job Priority: Constrained and Arbitrary-Deadline 5/12

- The idea presented by Baker is that
  if the job of task τ_k misses its deadline
  then the load in the interval must be at least: $m(1-\delta_k) + \delta_k$
  where $\delta_i = C_i / \min(D_i, T_i)$
- 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 $m(1-\delta_k) + \delta_k$

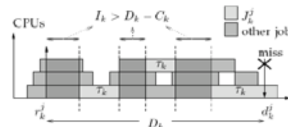


Global Fixed Job Priority: Constrained and Arbitrary-Deadline 6/12

- This gives the following sufficient schedulability test:
- *A constrained-deadline task set is schedulable under pre-emptive global EDF scheduling if for every task τ_k :*

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

- Where β_i is an upper bound on the processor load due to task τ_i for any problem window relating to τ_k



Global Fixed Job Priority: Constrained and Arbitrary-Deadline 7/12

- Baker extended this approach to sporadic tasks with arbitrary deadlines
  - The complexity of Baker's test is $O(n^3)$ in the number of tasks
- The basic strategy proposed by Baker is a seminal result which has been built upon by a significant thread of subsequent research
- Bertogna et al. 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
  - The complexity of this test is $O(n^2)$ in the number of tasks
- Baruah derived a sufficient schedulability test for global EDF scheduling of sporadic task sets with constrained deadlines
  - Uses the same basic approach as Baker
  - 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
  - The test limits the number of tasks that are counted as causing carry-in interference to m-1

Global Fixed Job Priority: Constrained and Arbitrary-Deadline 8/12

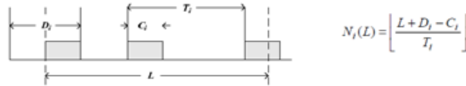
- In 2008, building on their previous work, Baruah and Baker derived a further sufficient test for global EDF scheduling of sporadic task sets with constrained deadlines
  - Limits the number of tasks that are counted as causing carry-in interference
- This processor load based test is given by:
  $load(\tau) \leq \mu - (\lceil \mu \rceil - 1)\, \delta_{max}$
- where $\mu = m - (m-1)\delta_{max}$
- Baruah and Baker showed that this sufficient test, combined with global EDF scheduling, has a resource augmentation or speedup factor of:

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

- Baruah and Baker extended this study to sporadic task sets with arbitrary deadlines, showing that this test still applies

Global Fixed Job Priority: Constrained and Arbitrary-Deadline 9/12

- Bertogna et al. (2008) presented a schedulability test for sporadic task sets with constrained deadlines
  - that is valid for any work conserving algorithm
- Based on a consideration of the densest possible packing of interfering jobs in the problem window
- Wi (L) is an upper bound on the workload of task τ_i in an interval of length L: $W_i(L) = N_i(L)\, C_i + \min(C_i\, , L + D_i - C_i - N_i(L)\, T_i\,)$
  - where $N_i(L)$ is the maximum number of jobs of task τ_i that contribute all of their execution time in the interval



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

Global Fixed Job Priority: Constrained and Arbitrary-Deadline 10/12

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

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

- Bertogna et al. extended this test to the specific cases of global EDF
- Bertogna et al. further extended their approach via an iterative schedulability test that calculates the slack for each task
  - This approach is also applicable to any work-conserving algorithm
  - It was also specialised for global EDF

Global Fixed Job Priority: Constrained and Arbitrary-Deadline 11/12

- Baruah and Fisher (2007) derived the following sufficient test for jobs of sporadic tasks scheduled by global EDF
  - *load*(τ , j) is the processor load due to all jobs with higher priority than job j
- $load(\tau, j) \leq (1/(1+K))\ .\ (m - (m - 1)C_j\ /\ D_j\ )$
  - where K is the largest ratio between task deadlines
- As K may potentially take any value
  - this test does not have a finite resource augmentation factor

Global Fixed Job Priority: Constrained and Arbitrary-Deadline 12/12

- In 2008, Bonifaci et al. derived a sufficient schedulability test for global EDF scheduling of sporadic task sets with arbitrary deadlines which has a speedup factor of ($2 + 1/m + \varepsilon$) for arbitrarily small ε
- Recall that Phillips et al. showed that global EDF requires m processors of speed ($2{+}1/m$) in order to schedule all task sets that are feasible on m processors of unit speed
- The schedulability test introduced by Bonifaci et al. and extended by Baruah et al. has the property that there are no task sets 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
- No schedulability test exists for global EDF that requires a smaller speedup factor

Global Fixed Task Priority

- 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

Global Fixed Task Priority: Implicit Deadline 1/5

- The seminal work of Dhall and Liu in 1978 also considered global scheduling of periodic task sets with implicit deadlines on m processors
- They showed that the utilization bound for global RM scheduling is $1{+}\varepsilon$
  - for arbitrarily small ε
- This occurs when there are
  m tasks with short periods/deadlines and infinitesimal utilization and
  one task with a longer period/deadline and utilization that approaches 1

Global Fixed Task Priority: Implicit Deadline 2/5

- In 2001, B. Andersson et al. showed that any periodic task set with implicit deadlines can be scheduled using global RM scheduling provided that:
  $u_{\max} \le m / (3m - 2)$ and $u_{sum} \le m^2 / (3m - 1)$
- Com m $\rightarrow\infty$
  $u_{\max} \le 1/3$ and $u_{sum} \le m/3$

Global Fixed Task Priority: Implicit Deadline 3/5

- B. Andersson et al. also proposed the RM-US[ς] algorithm
- It gives the highest priority to tasks with utilization greater than the threshold ς (with ties broken arbitrarily)
  - and otherwise assigns priorities in RM order
- RM-US[$m/(3m-2)$] has a utilization bound of:
  $U_{RM-US}[m/(3m-2)] = m^2 / (3m - 1)$
- In 2002, Lundberg showed that that setting the threshold used in RM-US[ς] to 0.375 results in the following utilization bound which is the maximum possible bound for this algorithm:
  $U_{RM-US}[0.375] \approx 0.375\, m$

Global Fixed Task Priority: Implicit Deadline 4/5

- In 2003, B. Andersson and Jonsson showed
  - for periodic task sets with implicit deadlines
  - where priorities are defined as a scale invariant function of task periods and worst-case execution times
- the maximum utilization bound for any global fixed task priority scheduling algorithm:
  $U_{OPT} \le (2 - 1)m \approx 0.41m$
- In 2005, Bertogna et al. tightened the bound for global RM scheduling to:

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

Global Fixed Task Priority: Implicit Deadline 5/5

- In 2008, B. Andersson proposed a ‘slack monotonic’ algorithm (SM-US)
  - where priorities are ordered according to the slack of each task given by $T_i - C_i$
  - otherwise works in the same way as RM-US
- SM-US[2/(3+5)] has a utilization bound of:

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

  - sporadic task sets with implicit deadlines

Global Fixed Task Priority: Constrained and Arbitrary Deadline 1/3

- In 2005, Bertogna proved the following density bound for global DM scheduling of sporadic task sets with constrained deadlines:
  $\delta_{sum} \le m/2\ (1 - \delta_{max}) + \delta_{max}$
- Bertogna et al. proposed the DM-DS[ς] algorithm
  - It gives the highest priority to at most m-1 tasks with density greater than ς
  - and otherwise assigns priorities in Deadline Monotonic priority order
- Under DM-DS[ς] a taskset is schedulable provided that:

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

- Where ψ is the number of ‘privileged’ tasks with density higher than the threshold, and $\delta^{(m)}$ is the density of the mth highest density task
- They proved the sufficient test for DM-DS[1/3]: $\delta_{sum} \le (m+1)/3$

Global Fixed Task Priority: Constrained and Arbitrary Deadline 2/3

- Guan et al. observed that the worst-case response time for a job of task τk 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. improved a previous response test from Bertogna and Cirinei as follows

Global Fixed Task Priority: Constrained and Arbitrary Deadline 3/3

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

- where *Max*(*k*,*m*-1) is the subset of tasks with higher priorities than τk, with the *m*-1 largest values of $I_i^{DIFF}(R_k^{UB})$

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

$$\bar{I}_i^{NC}(R_k^{UB}) = \min(W_i^{NC}(R_k^{UB}), R_k^{UB} - C_k + 1)$$

$$W_i^{NC}(L) = N_i^{NC}(L)C_i + \min(C_i, L - N_i^{NC}(L)T_i) \qquad N_i^{NC}(L) = \left\lfloor \frac{L}{T_i} \right\rfloor$$

Global dynamic priority scheduling

- A number of these algorithms are known to be optimal for periodic task sets with implicit deadlines
  - Pfair and its variants PD, PD^2, ERFair, BF
  - SA
  - LLREF
- However, it is known that there are no optimal online (non clairvoyant) algorithms for the pre-emptive scheduling of sporadic task sets on multiprocessors
- 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

Global Dynamic Priority: Proportionate Fairness 1/6

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

Global Dynamic Priority: Proportionate Fairness 2/6

- Baruah et al. showed that the Pfair algorithm is optimal for periodic task sets with implicit deadlines, with a utilisation bound of:
  $U_{PFAIR} = m$
- 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, PD, and PD^2

Global Dynamic Priority: Proportionate Fairness 3/6

- The amount of execution time that should ideally have been allocated to task τ_i by time *t* is $t.u_i$
- The $lag(\tau_i, t)$ is given by the amount of execution time that should ideally have been allocated to task τ_i by time *t* less the processing time actually allocated
- The Pfair algorithm ensures that the $lag(\tau_i, t)$ 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 and PD_2 improve on the efficiency of Pfair by separating tasks into groups of *heavy* ($u_i > 0.5$) and *light* tasks

Global Dynamic Priority: Proportionate Fairness 4/6

- In 2000, J. Anderson and Srinivasan extended the PFair approach to sporadic task sets
- The EPDF (*earliest pseudo-deadline first*) algorithm, a variant of PD, is optimal for sporadic task sets
  - With implicit deadlines
  - executing on 2 processors
- But is not optimal for more than 2 processors

Global Dynamic Priority: Proportionate Fairness 5/6

- In 2003, Zhu et al. 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
  - BF ensures only that proportionate progress is made on all tasks at period boundaries, but not at other times
- BF is an optimal algorithm for periodic task sets with implicit deadlines
- The number of scheduling points is typically 25-50% of the number required for PD

Global Dynamic Priority: Proportionate Fairness 6/6

- In 2005, Holman and J. Anderson implemented Pfair scheduling on a symmetric multiprocessor
- The synchronised re-scheduling of all processors every time quanta caused significant bus contention due to data being re-loaded into cache
- Holman and J. Anderson developed a variant of Pfair
  - It 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

Global Dynamic Priority: SA

- In 1997, Khemka and Shyamasundar developed an optimal algorithm for periodic task sets with implicit deadlines called SA
- This algorithm takes at most $O(n + m + 1)H(\tau) / GCD(\tau)$ operations to build a schedule
  - $H(\tau)$ is the least common multiple of task periods
  - $GCD(\tau)$ 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

Global Dynamic Priority: LLREF 1/3

- In 2006, Cho et al. introduced the LLREF algorithm
  - optimal for periodic task sets with implicit deadlines

- LLREF divides the timeline into sections separated by normal scheduling events
- At the start of each section, *m* tasks are selected to execute on the basis of *largest local remaining execution time first* (LLREF)
- Local remaining execution time for task τ_i at the start of section *k* (length t^k_f)
  - the amount of execution time that the task would be allocated during that section in a fluid schedule $t^k_f u_i$
- The local remaining execution time decrements as a task executes during the section

- LLREF gives rise to additional scheduling events
  - when a running task completes its local execution time
  - or a non-running task reaches a state where it has no local laxity

Global Dynamic Priority: LLREF 2/3

- In 2008, Funaoka et al. extended the LLREF
- By
  - apportioning processing time that would otherwise be unused among the tasks reapportioning processing time when a task completes earlier than expected
- It creates a work-conserving algorithm

- Funaoka et al. showed that for task set utilizations below 100% this approach results in significantly fewer pre-emptions than LLREF

Global Dynamic Priority: LLREF 3/3

- In 2009, Funk and Nadadur extended the LLREF approach, forming the LRE-TL algorithm
- Within each section, there is no need to select tasks for execution based on largest local remaining execution time
  - 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 task sets
- They proved that it is optimal (utilization bound of 100%) for sporadic task sets with implicit deadlines

Global Dynamic Priority: EDZL 1/2

- In 1994, Lee introduced the Earliest Deadline until Zero Laxity (EDZL) algorithm
- It dominates global EDF scheduling
- 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

Global Dynamic Priority: EDZL 2/2

- In 2007, Cirinei and Baker provided a sufficient schedulability test:
- *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:*

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

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

- In 2008, Baker et al refined the sufficient test for EDZL:

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

Hybrid Approaches

- Depending on the hardware architecture, the overheads incurred by global scheduling can potentially be very high
  - Migration of jobs 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
- In fully partitioned approaches the available processing capacity can become fragmented
  - Although in total a large amount of capacity is unused, no single processor has sufficient capacity remaining to receive further tasks
  - The maximum utilization bound is just 50% of the total processing capacity
- Hybrid approaches combines elements of both partitioned and global scheduling

Hybrid Approaches: Semi-Partitioned Approaches 1/9

- In 2006, B. Andersson and Tovar introduced EKG
  - an approach to scheduling periodic task sets with implicit deadlines
  - based on partitioned scheduling
  - But splitting some tasks into two components that execute at different times on different processors
- The utilisation bound for EKG depends on the parameter *k*
  - used to control division of tasks into groups of heavy and light tasks
- The utilization bound for EKG is given by:
  - $U_{EKG} = k / (k+1)$ when $k < m$
  - $U_{EKG} = 1$ when $k = m$
- The average number of preemptions per job over the hyperperiod is bounded by $2k$
- Choosing a value of $k = 2$, gives
  - a utilization bound of 66%
  - at most an average of 4 pre-emptions per job

Hybrid Approaches: Semi-Partitioned Approaches 2/9

- In 2008, B. Andersson and Bletsas 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. later extended this approach to tasksets with arbitrary deadlines
- First-fit and next-fit are not good allocation strategies when task splitting is employed
- 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
  - the number of preemptions is increased if the time slots are short

Hybrid Approaches: Semi-Partitioned Approaches 3/9

- In the implicit deadline case, B. Andersson and Bletsas showed that this approach has a utilisation bound of:
  $U = 4(\sqrt{\delta(\delta+1)} - \delta) - 1$
- where δ effectively defines the slot length: (T_{min} / δ)
- This utilisation bound equates to approximately 88% for $\delta = 4$
- The number of additional pre-emptions in an interval of length *t* is given by: $3\delta \lceil t / T_{min} \rceil + 2$

Hybrid Approaches: Semi-Partitioned Approaches 4/9

- In 2009, Bletsas and B. Andersson developed an alternative approach based on the concept of 'notional processors'
- 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
  - 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
- It has a utilisation bound of at least 66.6% for task sets with implicit deadlines
- The number of additional preemption is: $\left(2m + \left\lceil \frac{m}{3} \right\rceil\right)\left\lceil \frac{t}{S} \right\rceil$
  - S is the minimum period of any task on the processor

Hybrid Approaches: Semi-Partitioned Approaches 5/9

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

Hybrid Approaches: Semi-Partitioned Approaches 6/9

- In 2008, Kato and Yamasaki presented EDDP, 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
- It 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
- EDDP has a utilization bound of 65% for periodic task sets with implicit deadlines
- Requires less context switches than EDF due to the placement strategy for heavy tasks
- Kato and Yamasaki also extended this approach to fixed task priority scheduling
  - RMDP algorithm has a utilization bound of 50%.

Hybrid Approaches: Semi-Partitioned Approaches 7/9

- In 2009, Kato et al. developed a semipartitioning algorithm called DM-PM (Deadline-Monotonic with Priority Migration)
  - applicable to sporadic tasksets, and using fixed priority scheduling
- DM-PM strictly dominates fully partitioned fixed task priority approaches
  - tasks are only permitted to migrate if they won't fit on any single processor
- Tasks chosen for migration are assigned the highest priority
  - portions of their execution time assigned to processors
  - 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
  - Execution begins on the next processor once the portion assigned to the previous processor completes
- DM-PM has a utilization bound of 50% for task sets with implicit deadlines
- Kato et al. extended the same basic approach to EDF scheduling
  - EDF-WM algorithm (EDF with Window constrained Migration)

Hybrid Approaches: Semi-Partitioned Approaches 8/9

- In 2009, Lakshmanan et al. developed a semipartitioning method based on fixed priority scheduling of sporadic tasksets with implicit or constrained deadlines
- 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

Hybrid Approaches: Semi-Partitioned Approaches 9/9

- Lakshmanan et al. 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), this bound increases to 65%
- 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 for single processor systems, without the restriction on individual task utilisation

Hybrid Approaches: Clustering 1/3

- Clustering can be thought of as a form of partitioning
- Clusters effectively form a smaller number of faster processors to which tasks are allocated
- Thus capacity fragmentation is less of an issue than partitioned approaches
- 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 worstcase execution time

Hybrid Approaches: Clustering 2/3

- In 2008, Shin et al. derived schedulability analysis where tasks are
  - Allocated to clusters of processors
  - 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
- The algorithm proposed was shown to be optimal for task sets with implicit deadlines
- The maximum number of preemptions is m-1 in an interval equal to the GCD (Greatest Common Divisor) of the task periods
  - In practice, this number of context switches can be prohibitive

Hybrid Approaches: Clustering 3/3

- In 2008, Leontyev and J. Anderson developed a container-based hierarchical scheduling scheme for multiprocessor systems executing both hard and soft real-time tasks
- Each container is allocated a specific bandwidth
  - This bandwidth is provided by a periodic server
- The tardiness of soft real-time tasks can be bounded in this model
  - without any loss of utilization
- Utilization loss does occur when hard-real-time tasks are included
  - This loss is small provided that the utilization of hard real-time tasks represents a small fraction of the total

Resource Sharing

- 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
- Simpson's four-slot mechanism preserves independence of execution
- This mechanism requires memory space for four copies of the data
- For more complex cases
  - with multiple writers
  - access to other types of shared object, e.g. registers in hardware peripherals
- then mutual exclusion is required

Resource Sharing: Partitioned Scheduling 1/4

- In 1988 Rajkumar et al introduced a multiprocessor variant of the Priority Ceiling Protocol called MPCP
- It is applicable to partitioned systems using fixed priorities
- 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
  - the task at the head of the queue waiting on it is resumed
  - it executes at the ceiling priority of the resource

Resource Sharing: Partitioned Scheduling 2/4

- Allowing low priority tasks to execute while a higher priority task on the same processor is blocked on a global resource permits further priority inversion
- The low priority task can attempt to access another locked global resource with a higher ceiling and
  - it can 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
  - nesting of local and global critical sections is not permitted
- MPCP provides a bounded blocking time
  - a sufficient schedulability test based on the utilisation bound of Liu and Layland
- The blocking factor is made up of five different components

Resource Sharing: Partitioned Scheduling 3/4

- In 1994 Chen et al. described a further variant of PCP called MDPCP
- 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

Resource Sharing: Partitioned Scheduling 4/4

- In 2001, Gai et al introduced the MSRP protocol based on SRP
- MSRP is applicable to partitioned systems using either fixed priorities or EDF
- When a task is blocked on a global resource under MSRP
  - it busy waits and is not preemptable (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
  - A 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
  - it is significantly simpler to implement than MPCP

Resource Sharing: Global Scheduling 1/6

- In 2006, Devi et al considered shared resources under global EDF
- They suggested two simple approaches for short non-nested accesses to shared data structures:

  Spin-based queue locks and

  lock-free synchronisation

Resource Sharing: Global Scheduling 2/6

- 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
- The spin queue grants access to resources in FIFO order
- 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

Resource Sharing: Global Scheduling 3/6

- With lock-free synchronization
  operations on shared resources (data structures) are implemented as “retry-loops”
- Operations are opportunistically attempted and if there is contention, then they are retried until they are successful

Resource Sharing: Global Scheduling 4/6

- Devi et al. 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 suggests that
  the total overheads of spin-based queue locks
  are significantly less than that of lock-free synchronisation

Resource Sharing: Global Scheduling 5/6

- In 2007, Block et al. introduced the flexible multiprocessor locking protocol (FMLP)
- 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
  - the job currently accessing the resource inherits the priority of the highest priority job in the queue
- FMLP avoids 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 does not requires tasks accessing nested resources to be in the same processor
  - as is the case with MSRP
- FMLP optimises the simple case of non-nested access to short resources

Resource Sharing: Global Scheduling 6/6

- In 2008, Brandenburg et al. examined the relative performance of blocking and non-blocking approaches to accessing shared resources
- The blocking approaches used FMLP and considered both
  - Spinning (busy-waiting)
  - Suspending
- The non-blocking approaches considered were *lock-freedom* and *wait-freedom*
- Nonblocking approaches are preferable for small and simple resource objects
- 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

Empirical Investigations: Test Performance 1/7

- The most commonly used metric is the number of randomly generated task sets that are deemed schedulable
- 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
- Utilization/density based tests, and speedup factors are useful performance indicators
  - But they focus heavily on specific pathological tasksets
- 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

Empirical Investigations: Test Performance 2/7

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

Empirical Investigations: Test Performance 3/7

- In 2005, Baker 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 task sets 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

Empirical Investigations: Test Performance 4/7

- In 2007, considering global scheduling algorithms, Bertogna showed that the iterative response time test for global FP scheduling outperformed all other tests for global FP and global EDF and also similar tests for EDZL
- 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

Empirical Investigations: Test Performance 5/7

- In 2009, Bertogna 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

Empirical Investigations: Test Performance 6/7

- 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 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

Empirical Investigations: Test Performance 7/7

- In 2009, Davis and Burns showed that, in global FP scheduling, the number of randomly generated task sets deemed schedulable using the schedulability tests of Bertogna et al. [51] is significantly increased by using Audsley's optimal priority assignment policy rather than Deadline Monotonic priority assignment
- Although optimal for uniprocessor systems, DM 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 DM

Empirical Investigations: Measurements 1/2

- In 2008, Brandenburg et al. 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
- 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

Empirical Investigations: Measurements 2/2

- 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

Open Issues: Limits on Processor Utilization

- Fixed job-priority and partitioning algorithms are in the worst-case capable of utilizing only 50% of the available processing resource
- Global dynamic algorithms can in some cases utilize up to 100%
  - their overheads are typically prohibitive
- Further research is needed into minimally dynamic algorithms,
- Novel approaches to partitioning task execution that can increase guaranteed processing capability
  - without introducing significant overheads
- Hybrid approaches

Open Issues: Ineffective Schedulability Tests

- For the sporadic task model,
  empirical studies have shown that there is a large gap 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

- *"no finite collection of worst-case job arrival sequences has been identified for the global scheduling of sporadic task systems"*
  - Baruah 2007

Open Issues: Consideration of overheads

- Advanced hardware features
  - such as cache architecture
- have a large impact on the cost of migration
  - at the task and job level

- Recent experimental implementations on multiprocessor platforms show that the overheads of
  - Migration
  - context switching
  - run-queue manipulation
- are a key issue for multiprocessor scheduling
- Research into scheduling algorithms and analysis needs to take appropriate account of such overheads

Open Issues: Limited Task Models for Multiprocessors

- 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

- Collette et al. considers the limited job parallelism of each task defined by the rate at which it can execute on 1 to m processors
- Edmonds and Pruhs considers each task as a number of phases
  - each 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

Open Issues: Limited Policies for Access to Shared Resources

- In uniprocessor systems the Stack Resource Policy 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
  - non blocking approaches also perform well for simple resource accesses

- It 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

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

- 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

Summary

- Motivation
- Introduction
- Taxonomy of Multiprocessor Scheduling Algorithms
- Performance Metrics
- Anomalies
- Partitioned Scheduling
- Global Scheduling
- Hybrid Approaches
- Resource Sharing
- Empirical Investigations
- Open Issues

Notation 1/3

- 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} Maximal utilisation of any task in the taskset
- u_{sum} Taskset utilization

Notation 2/3

- δ_i Density of task τ_i, $\delta_i = C_i / \min(D_i, T_i)$
- δ_{max} Maximal 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 thehyperperiod of the taskset)
- m Number of processors
- t Time
- $h(t)$ Processor demand in the interval $[0,t)$

Notation 3/3

- 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*