Event Based Scheduling of Real-Time Multicore Systems

An Elaboration of my talk for the KvBK Seminar

Frederik Völkel
Friedrich-Alexander-Universiät
Erlangen-Nürnberg

ABSTRACT

This elaboration of my talk for the "Konzepte von Betriebssystem-Komponenten"-Seminar provides an introduction to hard real-time scheduling algorithms for homogeneous multiprocessor systems. It defines and introduces some real-time scheduling specific terms and definitions. Especially the classification of real-time scheduling algorithms, schedulability, feasibility and optimality are covered. It gives an overview over partitioned, global an approaches to real-time scheduling and outlines their advantages and disadvantages. In the end some performance metrics are mentioned and some topics for further research are presented.

1. INTRODUCTION

Since single processors won't become much faster these days multiprocessor systems are essential. This development requires research in scheduling algorithms for real-time multi processor systems. Using more then one processors causes new problems to scheduling algorithms. The main problem is basically on witch processor should a task run, should is be possible to migrate a task to another processor and in witch case should the scheduler migrate, as migrating task causes overhead on bus-systems and caches.

2. OVERVIEW

In this section different classes of Scheduling Algorithms are introduced and some definitions are made some necessary terms for talking about real-time scheduling.

2.1 Classification of Mulitprocessor Scheduling Algorithms

In real-time multiprocessor systems there are two main issues that a scheduling algorithm must tackle. Where should a task be executed(allocation problem) and when(priority problem) [6]? The allocation problem divides into three different approaches [9].

1. No migration. Tasks are assigned to one processor and can not be migrated.
2. Task-level migration. Migration is allowed on task-level.
3. Job-level migration. Migration is allowed on job-level.

Scheduling algorithms where migration is allowed are also called global algorithms and algorithms with no migration are called partitioned algorithms. A combinations of both are hybrid algorithms.

1. Fixed task priority. Tasks have one static priority applied to all of there jobs.
2. Fixed job priority. Each job has one static priority.
3. Dynamic priority. Priorities of jobs may change during execution time.

2.2 Schedulability

A task or a taskset can be schedulable. A task is called schedulable if a scheduling algorithm can schedule the task with its worst-case response time without missing the tasks deadline. Furthermore a taskset is called schedulable when the amount of its tasks are schedulable. You can proof schedulability with schedulability tests. A sufficient test only calls schedulable tasksets schedulable. A necessary test don't have to be sufficient but if a taskset does not pass the test it is definitely not schdulable. A exact test is sufficient and necessary [9].

2.3 Feasibility

A taskset is called feasible if there is an algorithm witch can schedule any combination of tasks out of this taskset [9].

For example showing feasibility for implicit-deadline periodic tasksets is quite is quite easy. Such a taskset is feasible if the following condition is true [14].

$$u_{sum} \leq m. \tag{1}$$

u_{sum} is the sum of the utilization(4.1) of every task in that taskset. And m is the number of processors. There are also some tighter feasibility conditions but the formulas are a bit more complex.

2.4 Optimality

A scheduling algorithm is called optimal when it can schedule all of the feasible tasksets of a specific task model [9]. In other words an algorithm is optimal if it can schedule all tasksets witch could be scheduled by any other algorithm [9].

Finding optimal algorithms for multiprocessor systems is very difficult and if the jobs are arbitrary and there if there

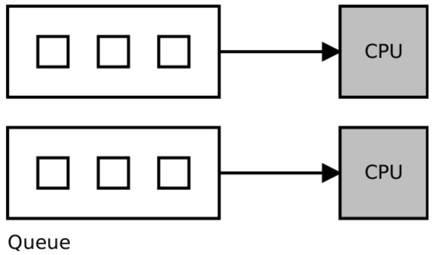

Figure 1: Partitioned scheduling

is more the one deadline it is impossible [13]. But there are optimal algorithms for periodic tasksets with implicit deadlines; see section 4.2.

3. PARTITIONED SCHEDULING

This section provides an overview over Partitioned Scheduling algorithms. It outlines advantages and disadvantages. Partitioned scheduling algorithms doesn't allow task or job migration from one processor to another. This means you need a strategy for assigning tasks to different processors and a scheduling strategy on the CPUs.

Assigning tasks to processors is the so called allocation problem. Witch in analogous to the bin packing problem, witch means how to assign n objects(tasks) of a specific weight(execution time) to m buckets(processors). Finding an optimal solution for the problem is NP-hard as Garey and Johnson showed in 1979 [11]. Witch means the problem can not be solved in polynomial time assuming $N \neq NP$.

The second problem is simply how to set the priorities to the various tasks and the one with the highest receives CPU time.

Partitioned scheduling has the following benefits and disadvantages.

- Overstepping a deadline can only affect tasks on the same CPU.
- Separated run queues are used for each processor; see figure 1.
- No migration costs.
- CPUs are often idle. In the worst cases up to 50% of the time [9].
- You need to find for every new taskset an new allocation to the processors.

The biggest advantage of partitioned scheduling is that after assigning the tasks to different processors you can simply apply uniprocessor scheduling algorithms [9]. On these algorithms is a lot of research done and often they are optimal. The biggest problem is in fact the allocation problem as it is NP-hard [11].

3.1 Allocation Problem

Next fit(NF) is the simplest bin packing algorithm it simply takes an object and puts it into a bin until it is full then it takes the next bin an fills it. This goes until every object is assigned to one bin. The complexity of NF is $O(n)$.

A better but slower algorithm is the first fit algorithm(FF). It is similar to NF but is checks before taking a new bin if there is another bin that fits the object. Therefore it has a complexity of $O(n * log(n))$. The best fit algorithm(BF) chooses the bin with the least capacity left that still fits the object. It has also a complexity of $O(n * log(n))$.

To improve these strategies you can sort the objects deceasing or increasing by various attributes. This is often used for constrained and arbitrary deadline tasksets.

There are a few more of those simple strategies for the bin packing problem but these are the most important ones.

The small tasks algorithm(ST) allocates tasks with similar harmonic periods to the same processor. This approach is particularly good for tasks with small utilization(4.1).

An algorithm witch combines 2 allocation strategies is the general task(GT) algorithm. It separates tasks into two groups. One group with tasks with a utilization greater then 1/3 and one with utilization less or equal to 1/3. Tasks from the first group are assigned with the ST algorithm and the ones from the second group are assigned by a FF algorithm. This algorithm has a complexity of $O(n)$ and is used for a more general tasksets [4].

3.2 Priority Assignment

There are three main strategies for priority assignment.

1. Rate monotonic (RM). The shortest period obtains the highest priority and the longest period obtains the lowest.
2. Deadline monotonic (DM). The shortest relative deadline obtains the highest priority and the longest obtains the lowest.
3. Earliest deadline first(EDF). The earliest absolute Deadline obtains the highest priority and the latest the lowest.

In the case of implicit tasksets(deadline = period) RM and DM is the same. EDF is dynamic priority assignment and RM and DM is fixed.

4. GLOBAL SCHEDULING

In this section I cover global scheduling algorithms with job-level migration, since this is the major class of global algorithms. I explicitly cover utilization based algorithms and proportionate fair algorithms. Allowing migration form one processor to another makes scheduling algorithms generally a lot more versatile but also more complex. For instance a global algorithm with dynamic priorities dominates [1] all other classes of scheduling algorithms.

In general, global approaches to real-time scheduling have the following advantages and disadvantages.

- Usually there are fewer context switches because a preemption will only occur when no other processor is idle [1].
- Effects of greater or less execution time then expected can be distributed to all other processors.
- No need for allocation algorithms as described in section 3.
- One single run queue for all processors; see figure 2. This results in a queue with many tasks in it and with

[1]An algorithm A dominates B when A can schedule all schedulable tasksets of B. But tasksets exists witch are schedulable by A and not by B

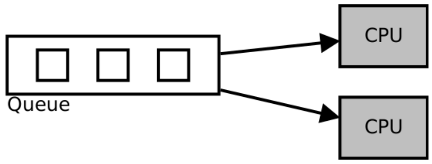

Figure 2: Global scheduling

many processors synchronizing the queue becomes ineffective.

The simplest way to do global scheduling is pick an algorithm like EDF and assign the m tasks with the highest priority to the m processor. But this approach doesn't perform well due to overheads when modifying a long queue with may processors. Brandenburg et al showed this on a system with 32 logical cores in 2008 [3]. For this reason better algorithms for multicore systems were developent as shown below.

4.1 Scheduling Based on Utilization

The utilization of a task is simply the worst execution time devided by the period of the task. Witch means how mutch time of a period needs a task in the worst case.

Utilization based scheduling works by dividing task in two groups based on their utilization. One group obtains the highest priority and the priorities of the other groups are determined by an algorithm such as EDF or RM priority assignment.

The <any priority assignment>-US algorithms assign the highest priority to tasks with a utilization greater then a specific threshold.

The EDF(k) algorithm assigns the highest priority to the k task with the highest utilization and uses EDF for the rest.

4.2 Proportionate Fair Algorithms

The proportionate fair algorithm(Pfair) splits time into quanta's of a specific length. Only at the beginning of each quanta the scheduler assigns tasks to processors. Each task obtains execution time in a quanta proportionate to its utilization. For example if the quanta's have a length of 2 and the utilization of the task is 0.5 the task obtains an execution time of 1. This algorithm is optimal for for periodic tasksets with implicit deadlines [2]. A special case of Pfair is the boundary fair(BF) algorithm witch simply sets the length of the quanta's equal to the period. BF is also optimal for periodic tasksets with implicit deadlines [15]. The problem of Pfair algorithm is, that at every quanta all processors must synchronize and scheduling decisions must be made. There are also a lot more context switches in comparison to other algorithms. In practice this causes so much overhead that a standard Pfair performs relatively poor [3].

For this reason a number of deviates of Pfair were developed. For example PD and PD^2 witch improves the efficiency of Pfair by dividing tasks into heavy and light tasks based on their utilization.

Holman and Anderson found out that with normal Pfair there is a lot of bus usage at the beginning of each time quanta. So they came up with the solution to stagger the quanta's which means to to shift the time quanta's between processors. With this enhancement the time quanta's begin on different times on each processor and the bus load will be reduced [12]. That the staggered Pfair performs in practice better then a normal Pfair is also shown by Brandenburg et al [3].

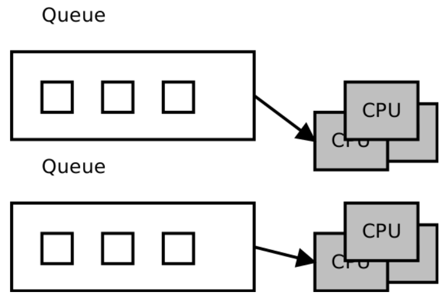

Figure 3: Clustering

5. HYBRID SCHEDULING

Due to the outlined disadvantages(section 3 and 4) of global and partitioned scheduling it is nearby to combine global and partitioned scheduling and eliminate some disadvantages. Hybrid algorithms approach in particular the utilization problem of partitioned scheduling and the migration costs of global algorithms.

One interesting hybrid solution is to combine processors to different clusters. This means to assign tasks to clusters beforehand like in partitioned scheduling but in the clusters a global algorithm is used; see figure 3. With such a solution it is possible to tackle for instance the migration cost problem by using shared caches within the clusters.

On large scale multi processor platforms such an algorithm performs better then any other algorithm as showed by Calandrino et al [5].

An other approach is to use a partitioned algorithm but allow one(usually the one with the highest priority) or a small number of tasks to migrate between processors. This approach is trying to improve the bad utilization of fully partitioned algorithms. In practice it shows that such a semi partitioned algorithm usually have better worst case utilizations [9].

6. PERFORMANCE METRICS

There are a few methods to compare the performance of real-time scheduling algorithms the following section will briefly outline three of them. Utilization bounds, approximation ratio and empirical measures.

6.1 Utilization Bounds

Utilization bounds are used for implicit-deadline tasksets. There are worst-case utilization bounds and best-cast utilization bounds. The first ones are usually the one of interest. A worst-case utilization bound basically means to put your system under full load but keeping the taskset schedulable. And the worst utilization you get is the worst-case utilization bound. In more formal words: "The worst-case utilization bound U_A for a scheduling algorithm A is defined as the minimum utilization of any implicit-deadline taskset that is only just schedulable according to algorithm A." [9]

6.2 Approximation Ratio

With an approximation ratio you can compare an algorithm with an optimal algorithm. In informal words the approximation ratio is setting the minimal number of processors needed to schedule a taskset with the optimal algorithm, into relationship with the minimal number of processors needed to schedule the same taskset with the other algorithm. Therefor a algorithm with an approximation ratio of one is optimal [9].

6.3 Resource Augmentation

Resource augmentation or also called speedup factor works similar than the approximation ratio but it does not use the number of processors to compare an algorithm with an optimal one, instead it uses the processing speed. To calculate a speedup factor of an algorithm A a optimal algorithm with m processors of speed 1 is compared with the algorithm A. The factor of witch to increase the speed of the processors to schedule all feasible tasksets of the optimal algorithm, with the algorithm A is the speedup factor [9].

6.4 Empirical Measures

To compare the effectiveness of various algorithms you can generate an number of random tasksets and test how many of them are schedulable with the different algorithms. This technique is easy to do does not take much time to do and is for the most cases sufficient enough. This makes this metric very important for the industry [9].

Another empirical measure is simulation of the schedules produced by various algorithms. A simulation enables to count migration and preemptions of of different algorithms and compare the algorithms in this aspect. Simulations aren't sufficient to show schedulability but they are good enough to show unschedulability [9].

7. ACTUAL STATE OF RESEARCH

Energy aware scheduling of real-time systems seams to be a topic these days as a few papers were published in 2015.

Also about fault-tolerant scheduling some papers were published in the last two years.

8. OPEN ISSUES

As mentioned in section 3 processors are in the worst case 50% of the time idle in partitioned fixed-job-priority scheduling. New ideas for algorithms witch increase utilization of processors without creating much scheduling overhead [9].

More research is needed in algorithms that consider the complex architecture of modern hardware, for their scheduling decisions. To approach this problem algorithms can limit the migration of task/jobs. In such algorithms further research is needed.

A lot more research is needed in algorithms witch are not based on uniprocessor algorithms. For example spitting task into various phases [10] or considering the work-limited job parallelism of each task defined by the rate at witch it can execute on 1 to m processors [9, 7, 8].

Also schedulability tests for sporadic task models is a research area with some problems to solve because there is a big gap between what existing tests showing is possible to schedule and what actually can be scheduled in practice [9].

9. REFERENCES

[1] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. In *Real-Time Computing Systems and Applications, 2000. Proceedings. Seventh International Conference on*, pages 337–346. IEEE, 2000.

[2] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[3] B. B. Brandenburg, J. M. Calandrino, and J. H. Anderson. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *Real-Time Systems Symposium, 2008*, pages 157–169. IEEE, 2008.

[4] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *Computers, IEEE Transactions on*, 44(12):1429–1442, 1995.

[5] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on*, pages 247–258. IEEE, 2007.

[6] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. *Handbook on scheduling algorithms, methods, and models*, pages 30–1, 2004.

[7] S. Collette, L. Cucu, and J. Goossens. Algorithm and complexity for the global scheduling of sporadic tasks on multiprocessors with work-limited parallelism. *RTNS'07*, page 123, 2007.

[8] S. Collette, L. Cucu, and J. Goossens. Integrating job parallelism in real-time scheduling theory. *Information Processing Letters*, 106(5):180–187, 2008.

[9] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.

[10] J. Edmonds and K. Pruhs. Scalably scheduling processes with arbitrary speedup curves. *ACM Transactions on Algorithms (TALG)*, 8(3):28, 2012.

[11] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1979.

[12] P. Holman and J. H. Anderson. Adapting pfair scheduling for symmetric multiprocessors. *Journal of Embedded Computing*, 1(4):543–564, 2005.

[13] K. S. Hong and J. Y. Leung. On-line scheduling of real-time tasks. In *Real-Time Systems Symposium, 1988., Proceedings.*, pages 244–250. IEEE, 1988.

[14] W. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.

[15] D. Zhu, D. Mossé, and R. Melhem. Multiple-resource periodic scheduling problem: how much fairness is necessary? In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 142–151. IEEE, 2003.