Event Based Scheduling of Real-Time Multicore Systems

An Elaboration of my talk for the KvBK Seminar

Frederik Völkel
Friedrich-Alexander-Universität
Erlangen-Nürnberg

ABSTRACT

This elaboration 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. It gives an overview over partitioned, global and hybrid 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 doesn’t get 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 it 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 I introduce different classes of Scheduling Algorithms and I will define some necessary terms for talking about real-time scheduling.

2.1 Classification of Multitopper 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(prioritization problem) [5]? The allocation problem divides into three different approaches [8].

1. No migration. Tasks are assigned to one processor and cannot 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 is referred to as schedulable according to a given scheduling algorithm if its worst-case response time under that scheduling algorithm is less than or equal to its deadline. Similarly, a taskset is referred to as schedulable according to a given scheduling algorithm if all of its tasks are schedulable.” [8]

2.3 Feasibility

“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”[8].

For example showing feasibility for implicit-deadline periodic tasksets is quite easy. Such a taskset is feasible if the following condition is true [12].

\[ u_{sum} \leq m. \] (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 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.” [8]. Finding optimal algorithms for multiprocessor systems is very difficult and if the jobs are arbitrary and there if there is more the one deadline it is impossible [11]. 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 don’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 [10]. 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 gets 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.
- No migration costs.
- CPUs are often idle. In the worst cases up to 50% of the time [8].
- 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 [8]. 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 [10].

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 is 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. I has also a complexity of O(n * log(n)).

To improve these strategies you can sort the objects decreasing 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 I will leave it at that and continue with some more real time specific approaches.

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 which 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 [3].

3.2 Priority Assignment

There are three main strategies for priority assignment.

1. Rate monotonic (RM). The shortest period gets the highest priority and the longest period gets the lowest.

2. Deadline monotonic (DM). The shortest relative deadline gets the highest priority and the longest gets the lowest.

3. Earliest deadline first(EDF). The earliest absolute Deadline gets 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 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 pre-emption 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. This results in a queue with many tasks in it and with many processors synchronizing the queue gets ineffective.

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 much 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 gets 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..

\[\text{An algorithm } A \text{ dominates } B \text{ when } A \text{ can schedule all schedulable tasksets of } B. \text{ But tasksets exists witch are schedulable by } A \text{ and not by } B\]
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 gets 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 gets 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 which simply sets the length of the quanta’s equal to the period. BF is also optimal for periodic tasksets with implicit deadlines [13]. 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.

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

5. HYBRID SCHEDULING

Due to the outlined disadvantages (section 3 and 4) of global and partitioned scheduling it is nearly 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. With such a solution it is possible to tackle for instance the migration cost problem by using shared caches within the clusters.

Also on large scale multi processor platforms such an algorithm performs better than any other algorithm as showed by Calandrino et al [4].

6. PERFORMANCE METRICS

There are a few methods to compare the performance of real-time scheduling algorithms in the following section I 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. “The worst-case utilization bound for a scheduling algorithm is defined as the minimum utilization of any implicit-deadline taskset that is only just schedulable according to the algorithm.” [8]

6.2 Approximation Ratio

With an approximation ratio you can compare an algorithm with an optimal algorithm. In uniform 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.

6.3 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.

Another empirical measure is simulation of the schedules produced by the algorithms. Simulations aren’t sufficient to show schedulability but they are good enough to show unschedulability.

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 idle in partitioned fixed-job-priority scheduling. New ideas for algorithms which increase utilization of processors without creating much scheduling overhead. [8]

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 which are not based on uniprocessor algorithms. For example splitting task into various phases [9] or considering the work-limited job parallelism of each task defined by the rate at which it can execute on 1 to m processors [8, 6, 7].

9. REFERENCES


