Literature Review
Real-Time Systems EEL 4932ST

By: Michael Fillinger and Jared Spinks
1 Introduction

The scope of this paper is to review several papers focused around the subject of real-time operating systems in a multiprocessor environment. The reviews will consist of a summary of the paper as well as a discussion at the end. The following papers will be reviewed:

1. Brandenburg, Bjorn B., and James H. Anderson. "Optimality results for multiprocessor real-time locking."

2 Review of “Optimality Results for Multiprocessor Real-Time Locking”

In the paper “Optimality Results for Multiprocessor Real-Time Locking” by Bjorn B. Brandenburg and James H. Anderson, the topic of optimal locking protocols for real-time multiprocessor systems is discussed. The paper primarily focuses on the schedulers global earliest deadline first (G-EDF), partitioned earliest deadline first (P-EDF), and partitioned static-priority (P-SP) [Brandenburg et al, p. 1]. In the paper, priority-inheritance blocking is separated into 2 categories: suspension-oblivious priority inheritance blocking, and suspension-aware priority inheritance blocking [Brandenburg et al, p. 2]. In the case of suspension-oblivious blocking, the suspensions are treated as if they were normal computation (hence the "oblivious"), and in suspension-aware blocking, they are treated like actual suspensions. Also in the paper, the authors acknowledge that nested resources are not considered, but can be handled by another locking protocol.

The paper goes on to describe the the locking protocols based on suspension-aware or suspension-oblivious, and then from there it is further separated into global schedulers or partitioned schedulers. For the global suspension-oblivious priority inversion blocking, the paper considers the O(m) locking protocol (OMLP) [Brandenburg et al, p. 2]. To summarize, for each resource there are two queues. One of them is a queue where the tasks are arranged in order of their priority, called the priority queue. The other queue is a first-in-first-out (FIFO) queue. The FIFO queue is of no greater than length m (m denotes the number of processors). The idea behind combining both of these queues is to allow higher-priority jobs to still take precedence over lower-priority jobs, while simultaneously preventing lower-priority jobs from being continuously blocked. The authors then go on to present a proof that this approach is asymptotically optimal when considering G-EDF. The following image is of Figure 4 from page 5, showing an example of the approach described in the paper [Brandenburg et al, p. 5]:
For the partitioned portion of the suspension-oblivious priority inversion blocking, the authors propose another version of OMLP referred to as partitioned OMLP [Brandenburg et al, p. 6]. Instead of using just 2 queues per resource, the partitioned method uses a token system. Each processor has a virtual token associated with it, and each processor also has a priority queue to assign said token. The job that is at the front of the queue will receive the token if it is available, and once a job receives a token, it is placed into the global FIFO queue for the resource it needs. The authors then go on to present a proof that this blocking protocol is asymptotically optimal as well.

The next portion of the paper discusses optimality for suspension-aware priority inheritance blocking. The authors go on to show that OMLP does not ensure the same upper-bound for blocking in suspension-aware priority inversion blocking. In this section of the paper, the authors present proofs that the global and partitioned models are optimal under different locking protocols. For global, suspension-aware priority inversion blocking, the authors present a proof that the flexible multiprocessor locking protocol (FMLP) is asymptotically optimal [Brandenburg et al, p. 8]. Shortly afterwards, the authors then go on to present a proof that suspension-aware priority inversion blocking under simple, partitioned FIFO locking protocol (SPFP) is asymptotically optimal [Brandenburg et al, p. 9]. The following table, from page 10, shows the maximum priority inversion blocking bounds of different protocol and queue combinations [Brandenburg et al, p. 10]:

![Figure 4: Example showing the global OMLP under G-EDF for six tasks sharing one resource on m = 2 processors. J_6 issues a request at t_0 = 3, enters FQ_1 at t_1 = 7, and holds l_1 at t_2 = 8. Note that J_1 and J_2 enter FQ_1 immediately for lack of contention, and thus J_2’s request precedes J_4’s request in spite of J_4 having an earlier deadline. In contrast, J_4 and J_5 arrive and enqueue after J_6, but enter FQ_1 before J_6 due to their earlier deadlines and Rule G3. Similarly, J_6 acquires l_k before J_1, despite J_3’s earlier request.](image_url)
In conclusion, this paper is significant because it introduces a new locking protocol with tighter bounds than what was previously known. Research into the topic of optimal blocking protocols allows schedulability tests to more precisely determine whether or not a given task set can be scheduled. While other blocking protocols already exist, it is important to come up with new ways to determine the bounds of the possible blocking terms so that they may be less pessimistic, and therefore possibly allow more tasks to be scheduled safely on already existing systems, or allow currently existing systems to be run with possibly fewer processors.

3 Review of “Techniques for Multiprocessor Global Schedulability Analysis”

In the paper, “Techniques for Multiprocessor Global Schedulability Analysis” by Sanjoy Baruah, the topic of Multiprocessor Global Schedulability is discussed and a new approach that outshines the previous approach is demonstrated and proved. To begin, the paper discusses the current model, background, and approaches that are currently being used in multiprocessor global scheduling. The model used in the paper is a fully preemptive, sporadic task model. The paper only focused on constrained and implicit deadline task systems. It then explained some of the problems that researchers face when working with global systems. One of these is that earliest deadline first (EDF) suffers from the Dhall Effect. The Dhall effect is an anomaly in which low-utilization tasks can be unschedulable on multi-processor systems, regardless of the number of processors. Dhall effect occurs when using scheduling Rate Monotonically (RM), Deadline Monotonically (DM), and with EDF. This downside is what prompted researchers to create hybrid versions of EDF that implement the fixed job-priority approach. Another one of the problems pointed out by Baruah is that we no not currently know a worst-case arrival sequence for Global Multiprocessor EDF, unlike for partitioned approaches. For partitioned approaches we have a worst-case arrival sequence called the Synchronous Arrival Sequence, allowing us to test partitioned systems with the worst-case. Without a worst-case arrival sequence, it is impossible to create necessary and sufficient schedulability tests for Global systems.

The paper then compares Partitioned FJP versus Global Fixed Job Priority (FJP) scheduling, starting with the current state of simulation experiments. Currently known global scheduling techniques perform worse than partitioned, meaning that more task sets are found to be schedulable under partitioned
approaches. The paper then proves a theorem; “Global and partitioned FJP scheduling are in-comparable.” [Baruah et al, p. 5]. This means that given new techniques to schedule and test global FJP, we could discover that Global FJP scheduling actually performs far greater than Partitioned FJP scheduling.

Next, Baruah moves to discuss the known and accepted tests for global EDF schedulability analysis. The first of which is the density test, created by Liu and Layland. The test was originally created by studying global multiprocessor scheduling with implicit deadline task systems and then modified to work with constrained-deadline task systems. The density test is sufficient and is as follows:

\[ \delta \text{ sum}(\tau) \leq m - (m - 1) \delta \text{ max}(\tau) \]

The next test was designed by Baker and is referred to simply as baker’s test. This framework for this test was then used by Bertogna to create a very similar, but simpler and more effective test, referred to in the paper as the [BCL] test. Both of the tests use the same technique and line of reasoning, but differ slightly in some of the values for the terms. The tests take a task system that is non-schedulable under Global EDF and looks at a portion of jobs in which a deadline was missed. Assuming that the portion is a legal sequence, we deem the the instant of the deadline miss to be \( t_d \). The test then looks at the period from \([t_0, t_d)\), and find the upper bounds on the amount of work that can be done, finding the non-schedulability condition. When doing so, the user has to be sure to account for the work from both jobs that start execution in the period and those who carry-in their work from before the period. Both Baker’s test the the [BCL] test have different definitions for \( t_0 \) and for the bounds on the work required.

The author then points out the negatives of both Baker’s and the [BCL] test. The first such negative being that they are geared strictly towards the worst-case scenario, which is rather pessimistic. The tests assume that every tasks carries work over from before the interval. When this assumption is made in very large task systems, it creates a very large overhead that cannot go ignored. Secondly, the tests both have very short execution times that could have been sacrificed for a more accurate testing procedure as schedulability tests should not be concerned with obtaining fast execution times over accuracy.

Finally, the author presents a new Global EDF-schedulability test. This testing procedure has a pseudo-polynomial run time, looks at different intervals than the other tests, and puts bounds on the number of carry-in tasks that the interval has. The test identifies sufficient conditions for ensuring that the task cannot miss any deadlines and then checks these conditions for all tasks to insure that no deadlines are missed in the system.

Compared to the currently accepted tests, the author’s new test runs slower, which is by design. The new test is shown to be the only test that is optimal on uniprocessor systems. Furthermore, it outperforms the other tests, when used on tasks systems that have a far greater a number of tasks that processors and in which the parameters of tasks vary widely in their magnitudes.

In conclusion, this paper is important because it shows the gaps in multiprocessor scheduling in relation to global scheduling approaches. The author explains why there is a lack of attention from the community on this topic and the problems that are holding multiprocessing back. Furthermore, the paper exposes the flaws of the current schedulability tests and offers a way to circumvent them, by creating a new testing procedure. This new test works with more life-like systems and offers more accurate results at the cost of run-time.

4 Review of “A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms”

In the paper, “A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms” by John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah,
the pros and cons of scheduling independent, periodic real-time task systems on a multiprocessor are explored. It first explores the classical scheduling approaches that are used in scheduling periodic tasks system: partitioning and global. As mentioned in the summary for [2], using approaches that are optimal on uniprocessor systems, such as EDF, while using global scheduling, will cause the system to suffer from the Dhall Effect. However, in recent times, Pfair scheduling has shown that it is capable of scheduling periodic tasks on multiprocessor environments. The paper then switches to a brief glimpse into partitioning and it's own problems. One such problem is that it presents a NP-hard problem whenever trying to find the optimal assignment of tasks to the available processors. It also discusses that existence of systems that are only schedulable when the tasks are not partitioned.

Next, the paper dives into discussing two different parameters for defining scheduling algorithms: The complexity of the priority scheme used and the degree of migration that is allowed. The complexity of priority scheme has 3 possibilities: Static, Fixed Dynamic, and Fully Dynamic. Likewise, the Degree of Migration also has 3 possibilities; No migration, Migration Restricted to job boundaries, and Unrestricted Migration. With two parameters, each with 3 possibilities, there exists 9 different classes of scheduling algorithms. Notation for describing is (x,y)-restricted, x is the priority class and y is the migration class.

<table>
<thead>
<tr>
<th>3: full migration</th>
<th>(1,3)-restricted</th>
<th>(2,3)-restricted</th>
<th>(3,3)-restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td>2: restricted migration</td>
<td>(1,2)-restricted</td>
<td>(2,2)-restricted</td>
<td>(3,2)-restricted</td>
</tr>
<tr>
<td>1: partitioned</td>
<td>(1,1)-restricted</td>
<td>(2,1)-restricted</td>
<td>(3,1)-restricted</td>
</tr>
<tr>
<td>1: static</td>
<td>2: job-level dynamic</td>
<td>3: unrestricted dynamic</td>
<td></td>
</tr>
</tbody>
</table>

The paper then discusses the relationships between scheduling algorithms. The author explains a key point saying that, in general, the more generalized of an approach one uses, the higher that the overhead costs will be. This is unlike relationships between scheduling algorithms which are not always obvious. The 3 possible relationship types are equivalent, strictly more powerful, or incomparable.

Equivalent means that if a task system is feasible under a (x,y)-restricted class, than it has to also be feasible under a (w,z)-restricted class. Strictly more powerful means that any task feasible under a (x,y)-restricted class will also be feasible under a (w,z)-restricted class; however, there will be at least one example of a task system that will be feasible under a (w,z)-restricted class, but not under a (x,y)-restricted class. Finally, incomparable means that there is a least one task system that is feasible under a (x,y)-restricted class but not feasible under a (w,z)-restricted class, and vice versa.
This paper discusses important properties of scheduling algorithms and sorts them into classes that are useful and give an abundance of information. Furthermore, these classes make it easy to see the relationships between different algorithms that would normally be difficult to see. These relationships give us insight into what approaches would be the most appropriate for the task system that we are trying to schedule and into what type of overhead can be expected.

5 Review of "Efficient scheduling algorithms for real-time multiprocessor systems"

The paper, "Efficient scheduling algorithms for real-time multiprocessor systems" by Ramamritham, Krithi, John A. Stankovic, and P-F. Shiah, is an older paper specifically focused on a couple scheduling algorithms for multiprocessor, real-time environments. At the time that this paper was written, knowledge of how to deal with resource contention and knowledge of how to create schedulability tests for real time environments was limited. Also, the authors consider a system where tasks arrive dynamically, as opposed to statically, claiming that static schedulers are less applicable. As a result, the authors use pragmatic algorithms to schedule tasks.

The authors consider two models for their multiprocessor system, shared memory and local memory. The shared memory model is akin to a modern global scheduling algorithm, as the authors state that it works by having a shared global memory where the tasks are loaded in, and from there they are scheduled on the first available processor. The local memory model that the authors propose also seems similar to more modern partitioned scheduling methods. The authors state that each processor has its own local memory (as opposed to globally shared), and that the tasks are loaded onto a single processors memory, and from there can be scheduled to run on said processor. The authors then go on to define parameters such as arrival time, deadline, worst case processing time, and resource use.

The first algorithm, under a section titled “A Heuristic Scheduling Algorithm” [Ramamritham et al, p. 185], uses a function to try and map out a feasible schedule for a given task set. The authors describe the problem of finding a feasible schedule as a search problem, and solve the problem as a form of search tree. Essentially, the authors state algorithm navigates down a branch of the tree in search of a feasible
schedule. If it cannot find a schedule, it backtracks and tries again. The authors propose to limit the amount of backtracking or the amount of times the function can run to reduce overhead from the backtracking. The authors state that this algorithm has $O(n^2)$ time complexity [Ramamritham et al, p. 186]. The second algorithm is under the section “The Myopic Scheduling Algorithm” [Ramamritham et al, p. 186]. The idea behind this algorithm is to consider only a certain number of tasks in advance, so $O(n^2)$ becomes $O(nk)$, and by fixing $k$ to a constant value, further reducing the time complexity to $O(n)$ [Ramamritham et al, p. 186]. At the end of the paper, the authors discuss how the myopic algorithm performs as well or better than the heuristic algorithm through simulations the authors ran.

This paper shows that, despite a lack of knowledge on the optimal way to approach a real-time scheduling problem, an efficient solution still may be found. By approaching the problem in a novel manner, in this case by treating it as a search tree problem, new solutions can be found. These solutions may give future insight on how to approach these problems in a more efficient or possibly optimal way.

6 Conclusion

Throughout this paper, four other papers covering a broad range of topics relating to real-time systems in a multiprocessor environment were covered. As multiprocessor systems become ever more prevalent in our society, the need for efficient and optimal solutions grow. By reviewing and discussing the papers above, we hope to further our knowledge on the subject, and gain insight on ways to approach multiprocessor real-time systems.
References


