The Many Faces of Multi-Level Real-Time Scheduling

Prof. John A. Stankovic
Department of Computer Science
University of Massachusetts
Amherst, MA, 01003

Abstract

Many real-time scheduling algorithms use simplistic sets of assumptions and this limits their applicability in practice [6]. Often, a large real-time system has multiple scheduling algorithms and multi-level scheduling algorithms. Multiple algorithms arise when it is possible to partition the system into subsystems, each with their own algorithm. While this is a valuable approach, in this paper we focus on the issue of multi-level scheduling. Multi-level scheduling arises for various important reasons. In this invited presentation we discuss multi-level scheduling examples from a process/thread model, local/distributed scheduling, manufacturing, and multimedia.

1 Introduction

Complex real-time systems usually don't have a single scheduling algorithm due to the system size, the vastly different requirements of various sets of tasks, and the different metrics used for different functions or subsystems. Decomposing a real-time system into subsystems may generate different scheduling algorithms. For example, front-end processors may execute a relatively small set of periodic tasks to process sensor data. The scheduling algorithm in this front-end may be a cyclic scheduler or the rate monotonic algorithm. An avionics subsystem of an aircraft might use a sporadic server algorithm and the navigation system might employ yet another algorithm such as EDF. While decomposition gives rise to multiple algorithms, these algorithms are not multi-level. This invited presentation focuses on multi-level scheduling algorithms. In particular, we demonstrate the need for more complex real-time scheduling research by using four examples of multi-level scheduling: two from operating systems and two from applications. The four examples are: the process/thread model, local/distributed scheduling, manufacturing applications, and multimedia used in a real-time control context.

2 Process/Thread Model

Many operating systems are now supporting a model of execution where a process can have multiple threads executing in its address space. If the schedulable entity is the thread, then this is a single level scheduling approach. However, if the schedulable entity is the process, then a second level scheduling decision must be made as to which thread of the process to execute. To guarantee that deadlines are met, the designer must consider both process and thread scheduling. Because of the greater difficulty involved in guaranteeing performance with this multi-level scheduling over single level thread scheduling, many real-time systems may avoid using this approach. However, in certain applications there are advantages to the multi-level approach. New scheduling and analysis techniques that account for context switching, shared state, hidden interactions and slowdowns, deadlines, etc. need to be developed for this situation. In the associated invited presentation, further details on the issues and problems involved will be presented.

3 Local/Distributed Scheduling

Most complex real-time systems are distributed. One of the least developed areas of real-time scheduling is distributed scheduling. Most current results in this area deal with static real-time systems. However, many actual distributed real-time systems require dynamic scheduling. In one example of dynamic, distributed real-time scheduling, the Spring system [3] advocates a dynamic guarantee scheduling algorithm that operates on each local multiprocessor node. If a newly arriving task or set of tasks with a deadline cannot be scheduled locally, then the task may be passed to a second level distributed scheduling algorithm [1], if the deadline is far enough away. In general, many other types of interactions between local and distributed levels of scheduling could occur. In fact, it has been proposed by various researchers that a meta level be used to dynamically control the parameter settings and types and amounts of interactions between the local and distributed scheduling levels.

4 Manufacturing

Agile manufacturing is a very demanding application with real-time constraints of many types. For example, to support agile manufacturing [4], it is often necessary to support scheduling of tasks with hard and soft deadlines, precedence constraints, shared resources and multiprocessing requirements.

Beyond this, additional requirements are imposed by the fact that scheduling must occur at different levels of abstraction. At the higher level the system must deal with orders for products, resources which consist of parts and subcomponents to be automatically assembled – constrained by robots, floor space, cost, and expected profits. Decisions made at this level, may be handled by a real-time AI subsystem, which determines which of the incoming orders need to be carried out, computational resources permitting. Whether it was possible to carry out a specific order is determined by the scheduler at the lower level, where the system deals with the *computational resources* needed to move robots, assemble products, etc. It is necessary to implement both levels of scheduling with a feedback interface between these levels. For example, if the higher level decides to make certain products, the actual manufacturing floor may not be capable of performing these tasks in time. Such information supplied to the higher level scheduler improves performance of the system in choosing between alternatives.

Interesting research questions regarding multi-level scheduling in manufacturing include:

- how to partition the scheduling functionality between the high level (RTAI) planner and the system level scheduler (which is also a planner),
- what information should be passed back and forth between the levels, and
- how to pass information back and forth between the schedulers so as to get the best performance.

In developing an agile manufacturing testbed [4], we found it necessary to be able to hold resources across a set of processes. For example, a set of processes may require a common tool which cannot be shared with others. This adds an interesting scheduling complication not typically addressed by real-time scheduling algorithms. We also found that deadlines can sometimes be relaxed within a *deadline tolerance*. Interesting questions involve understanding the cumulative effect of missing the original deadline, but satisfying the tolerance factor.

5 Multimedia

A new area of research is real-time multimedia. Consider the need to support scheduling for multimedia in a command, control and communications, C^3, environment[5]. Multimedia aspects of C^3 applications normally permit varying levels of real-time performance and resource requirements. Hard real-time tasks of these applications on the other hand, require deterministic QOS guarantees and predictability at a much finer grained temporal level, and consequently there is less opportunity for any safe resource/computation-quality tradeoff for these tasks. The emphasis of the hard real-time task requirements is more on ensuring predictable execution of already guaranteed tasks even at the cost of potential under-utilization of the system. In comparison, multimedia tasks are more amenable to adaptive and flexible scheduling paradigms where some degree of determinism can be traded off to improve system utilization without violating the QOS requirements of a particular application. Given the different concerns of the two types of applications, the hypothesis is that it may be better to use different scheduling policies for the different classes of tasks.

For multi-level scheduling in this type of application, we propose using separate execution time windows for continuous multimedia tasks and hard real-time command and control tasks, and applying different scheduling disciplines within each window. The basic ideas are to have (i) an efficient on-line scheduling algorithm for multimedia tasks that doesn't necessarily require strict scheduling plans, (ii) a precise time line algorithm for the hard real-time tasks, and (iii) to carefully address how the two algorithms impact each other because of shared resources. Obviously, simply having two classes of tasks with their own algorithms is not novel. Novelty is required in developing solutions that focus on the *true* influence each has on the other, especially with respect to meeting timing constraints. As one part of this, we are working on a solution that creates the following resource model. Resources are either passive (models buffers, memory, data structures) or active (models cpus, communication processors, disk controllers). Resources are also either precise time line or percentage based, or both. For example, a cpu is an active resource which in some windows has a precise time line semantics and in other windows has a percentage semantics. The algorithms we are developing then schedule and allocate across sets of these resources depending on their semantics. Such an integrated resource model can depict the influence of one class of algorithms on the other, at least as far as resource contention is concerned. Developing a truly predictable runtime platform is the other main requirement.

The overall main research questions that need to be addressed in this multi-level scheduling approach are:

- What is the most appropriate choice of the scheduling algorithm for each window? Since there are resources common to both the windows, there are effects at some level of the decisions taken by one scheduling discipline on tasks in the other window and vice versa. So the choice of particular algorithms for the different windows cannot be addressed as independent design issues. Having a truly predictable runtime platform eases the task of dealing with the impact of the two classes of scheduling on each other.
- In this multi-level scheduling, a difficult research questions is how to set up the different windows - i.e., what is the appropriate base-level scheduling needed to efficiently schedule resources among the two window types. Choices range from statically allocating some fixed fraction of CPU time to each window type using a round-robin fashion, to more elaborate schemes which dynamically vary the window sizes depending on relative importance of different tasks and/or the current system state and current mix of applications in the system. The first approach is easier to implement and the cost of base-level runtime scheduling is small, but clearly, this static scheme suffers from inflexibility and poor adaptability, and may result in low utilization of system resources. The dynamic approach, on the other hand, is much more responsive to changes in system state and

allows higher resource utilization, but may be accompanied by a higher runtime scheduling cost. More research needs to be done in evaluating the correct paradigm.

- A related issue is how to handle sharing of resources among tasks in different window types or different instances of the same window type. Depending on resource types, there may be need to provide some task with exclusive access to a particular resource across window boundaries. Resources may be non-preemptable (i.e., any usage of the resource must be in a non-preemptive manner) or serially reusable (e.g., CPU). Tasks might request exclusive or non-exclusive access to some resources. There needs to be sharing of information among the different window types to ensure resource consistency. Some policies are also needed at window boundaries to enforce some degree of independence between different window types. The separation of CPU time itself is difficult, but modeling and sharing of resources among deterministic and probabilistic uses pose hard research questions.

In the hard real-time window, one choice is to use a Spring-like scheduling approach [2] which uses planning during admission control to perform integrated scheduling of CPU and other resources. This planning approach avoids conflicts over resources by scheduling competing tasks to execute in different time intervals. The current Spring algorithm schedules tasks by planning task executions onto a continuous and precise time line into the future. Modifications need to be made to the algorithm to impart to it the ability to lay out plans onto discrete windows of time into the future. Besides the Spring algorithm, more general purpose real-time scheduling algorithms such as EDF (Earliest Deadline First) can be used. Using EDF leads to less scheduling overhead because it doesn't plan out tasks taking their resources into consideration. However, the implicit on-demand resource allocation model allows random interactions between different tasks competing for the same resource. This leads to poorer understandability of system resource contention and the resultant lack of deterministic resource control may result in poor predictability especially under overload conditions.

In the multimedia window, less precise guarantee based scheduling algorithms can be chosen. For example, Round Robin, fixed priority, EDF or the latest "best" scheduling algorithm can be used. The issues mentioned above when discussing EDF for hard real-time tasks are relevant here also. In addition, an important issue is the modeling of resources. That is, resources used in a window by different tasks need to be statistically shared yet ensuring QOS guarantees associated with the resources. Another question is whether the scheduling algorithm should be preemptive or nonpreemptive. Preemptive scheduling allows better statistical multiplexing of resources, but on the other hand, management of the resources across window boundaries is difficult since a task may get preempted holding some resources at a window boundary. An important issue is to incorporate into the chosen algorithm the capabilities to take advantage of the flexible resource requirements of multimedia tasks, their interval-based guarantees, and to exploit the unique resource modeling that we are proposing. The standard algorithms like EDF, Round Robin etc. do not have these abilities and require suitable modifications before they can be used.

6 Summary

Real-time scheduling research has received considerable attention, especially in the past 10 years. However, many of the problems addressed are too simplistic for direct use in many systems. Even though it is difficult and less likely to be amenable to closed form solutions, it is still necessary to develop scheduling approaches (both single and multi-level) that are comprehensive and integrated. For example, the overall approach must be comprehensive enough to handle:

- preemptable and non-preemptable tasks,
- periodic and non-periodic tasks,
- tasks with multiple levels of importance (or value function),
- groups of tasks with a single deadline,
- end-to-end timing constraints,
- precedence constraints,
- communication requirements,
- resource requirements,
- placement constraints,
- fault tolerance needs,
- tight and loose deadlines,
- normal and overload conditions, and
- different metrics in the same system.

The solution must be integrated enough to handle the interfaces between:

- CPU scheduling and resource allocation,
- I/O scheduling and CPU scheduling,
- CPU scheduling and real-time communication scheduling,
- local and distributed scheduling,
- static scheduling of safety-critical tasks and dynamic scheduling of less critical tasks, e.g., multimedia tasks, and
- disparate levels of resources such as in the manufacturing example.

Acknowledgments

This work was supported, in part, by NSF IRI-9208920, IRI-9318971, and NSF CDA-8922572.

References

[1] K. Ramamritham, J. Stankovic, and W. Zhao, Distributed Scheduling of Tasks with Deadlines and Resource Requirements, Special Issue of *IEEE Transactions on Computers*, Vol. 38, No. 8, pp. 1110-1123, August 1989.

[2] K. Ramamritham, and J. Stankovic, Scheduling Strategies Adopted in Spring: An Overview, chapter in *Foundations of Real-Time Computing: Scheduling and Resource Management*, edited by Andre van Tilborg and Gary Koob, Kluwer Academic Publishers, pp. 277-306, 1991.

[3] J. Stankovic and K. Ramamritham, The Spring Kernel: A New Paradigm for Real-Time Systems, *IEEE Software*, Vol. 8, No. 3, pp. 62-72, May 1991.

[4] J. Stankovic, K. Ramamritham, and G. Zlokapa, Real-Time Platforms and Environments for Time Constrained Flexible Manufacturing, *Workshop on Real-Time Operating Systems and Software*, May 1994.

[5] J. Stankovic, Continuous and Multimedia OS Support In Real-Time Control Applications, *Fifth Workshop on Hot Topics in Operating Systems*, pp. 8-11, May 1995.

[6] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo, Implications of Classical Scheduling Results For Real-Time Systems, *IEEE Computer*, Vol. 28, No. 6, pp. 16-25, June 1995.