Rate Monotonic Theory

A literature survey by Nate Forman

Introduction

Nobody likes missing deadlines. That kind of behavior can do horrible things to a class grade or cause trouble in the office. However, when dealing with hard deadlines in real-time systems, the consequences can be much more severe. If a hard real-time system misses its deadline, it can mean safety problems for patients at hospitals, thousands of dollars of lost product at a factory, or even failure for a multi-million dollar space mission. These consequences make scheduling in hard real-time systems a very relevant area of study. This survey discusses rate monotonic theory (analysis and scheduling), a model that allows schedulability analysis for real-time systems.

Basic Premises

The term rate monotonic derives from a method of assigning priorities to a set of processes as a monotonic function of their rates. [4] While rate monotonic scheduling systems use rate monotonic theory for actually scheduling sets of tasks, rate monotonic analysis can be used on tasks scheduled by many different systems to reason about schedulablility. We say that a task is schedulable if the sum of its preemption, execution, and blocking is less than its deadline.[2] A system is schedulable if all tasks meet their deadlines. Rate monotonic analysis provides a mathematical and scientific model for reasoning about schedulability.

Assumptions

Reasoning with rate monotonic analysis requires the presence of the following assumptions [4]:

- Task switching is instantaneous.
- Tasks account for all execution time.
- Task interactions are not allowed.
- Tasks become ready to execute precisely at the beginning of their periods and relinquish the CPU only when execution is complete.
- Task deadlines are always at the start of the next period.
- Tasks with shorter periods are assigned higher priorities; the criticality of tasks is not considered.
- Task execution is always consistent with its rate monotonic priority: a lower priority task never executes when a higher priority task is ready to execute.

It is immediately obvious that some of these assumptions do not completely conform to actual systems. However, extensions to broaden these assumptions will be discussed later. The importance of these assumptions is that they allow reasoning with certainty about whether or not a set of tasks can be scheduled.

Benefits

Given certain information about a particular set of tasks, under rate monotonic conditions, one can evaluate certain tests to understand whether or not those tasks can all meet their deadlines in a real time system. Because these values are known at design time and are monotonic, any analysis and scheduling can be done statically. Static scheduling is one advantage that the industry has a strong preference for in hard real-time

applications. [4] This subsection will examine two schedulability tests that can be used under rate-monotonic assumptions.

Given the computation time, C_i, and period, T_i, for task i, its CPU utilization can be calculated with the following equation:

$$U_i = C_i/T_i$$

For rate monotonic scheduling, the processor utilization for n tasks has been shown to be the following:

$$U(n) = n(2^{1/n} - 1)$$

U(n) asymptotically converges to ln(2) or 69%, which is less efficient than some runtime schedulers such as earliest deadline, but again, there is a strong preference for static scheduling. [4] The utilization bound (UB) test allows schedulability analysis by comparing the calculated utilization for a set of tasks and comparing that total to the theoretical utilization for that number of tasks:

$$C_1/T_1 + \ldots + C_n/T_n <= U(n) = n(2^{1/n} - 1)$$

If this equality is satisfied, all of the tasks will always meet their deadlines. If the total utilization calculates to greater than 100%, the system will have scheduling problems. However, if the total utilization is between the utilization bound and 100%, the UB test is inconclusive and a more precise test must be used.

The response time (RT) test allows analysis of schedulability based upon the following theorem:

> For a set of independent periodic tasks, if each task meets its deadline with worst case task phasing, the deadline will always be met. [2]

The RT test requires computation of the response time of each task in the system. Based on the above theorem if each response time is less than its corresponding period, the system is schedulable. The following is the calculation for a_n or the response time of task i:

$$a_{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j \qquad\qquad a_0 = \sum_{j=1}^{i} C_j$$

The test terminates when $a_{n+1} = a_n$. The system is schedulable if each response time finishes before its deadline.

Extensions

The interplay between research and application has resulted in the extension of rate monotonic theory from its original form of scheduling independent periodic tasks to scheduling both periodic and aperiodic tasks with synchronization requirements and mode change requirements. [4] These extensions have greatly enhanced the usability of the model while maintaining its desirable property of allowing mathematical reasoning about schedulability. The following subsections discuss some of these extensions.

Sporadic Server

Most systems are not limited to periodic tasks that happen regularly and monotonically, but also include aperiodic tasks. These tasks can be included in the model by the addition of one or more aperiodic servers. An aperiodic server is a conceptual task that is endowed with an execution budget and a replenishment period. An aperiodic server will handle randomly arriving requests at its assigned priority (determined by the RM algorithm based on its replenishment period) as long as the budget is available. [4]

Several versions of the aperiodic server were developed before a suitable one was found. That algorithm is the sporadic server algorithm. The sporadic server, like its predecessors, allocates a computation budget and a replenishment period to the execution of aperiodic tasks. However, with the sporadic server, the aperiodic task budget is not replenished periodically, but is replenished only after a period in which it was completely consumed. This implementation avoids a subtle violation of the rate monotonic assumptions, known as the deferred execution effect, that troubled earlier versions with different replenishment techniques. With this violation eliminated, the sporadic server is an aperiodic server that is equivalent to any periodic task under the rate monotonic assumptions.

Priority Ceiling Protocol

The NASA Mars Lander project was subject to hard deadline failures due to a condition called unbounded priority inversion. The lander had a number of instruments that communicated by a 1553 bus. The lander ran two processes that were critical to the correct operation of these instruments. The information distribution task had third priority in the system and the bus-scheduling task had first priority. Both of these tasks checked to see if the other had completed successfully during the cycle. The distribution task shared a synchronized resource with the low priority ASI/MET (accelerometer, radio altimeter, meteorological science) task.

During execution, the ASI/MET task acquired the shared resource and was subsequently preempted by several medium priority tasks. The distribution task was blocked and failed to meet its hard deadline. The bus-scheduler detected the failure and

reset the system. This problem was eventually detected and solved by initiating priority inheritance, discussed hereafter. [3]

For rate monotonic analysis to apply to systems where tasks share resources, it must address the issue of unbounded priority inversion. Unbounded priority inversion, as shown above, occurs when a high priority task fails to meet a hard deadline due to being blocked by a low priority task that has acquired a shared resource. This low priority process continues to hold the resource because it is preempted repeatedly by medium priority processes.

Two properties are added to avoid this condition, priority inheritance and priority ceilings. When a task blocks the execution of higher priority tasks it inherits the highest priority level of all of the tasks it blocks. In addition, a critical section is allowed to be entered only if the critical section will execute at a priority level that is higher than the inherited priority levels of any preempted critical sections. A system with these two properties has bounded priority inversion and is free from mutual deadlock. [4]

With these properties and guarantees in place, rate monotonic analysis allows reasoning about schedulability in systems where tasks share resources. To incorporate blocking into this reasoning, terms for blocking must be added to the UB and RT test. For UB, B_i/T_i must be added for each task i, where B_i is the maximum blocking time experienced by task i. In the RT test, B_i must be added to each iteration of a_{n+1}. With these adjustments made, the tests can be used in the same ways that they were before.

Other Application to Non Rate Monotonic Domains

Other extensions have been added to rate monotonic analysis to make it more widely applicable. [2] discusses extensions that deal with context switching overhead and

preemption by fixed priority interrupt tasks. In addition, [4] discusses a protocol for mode changes, or the addition or deletion of tasks. These are all made possible with either added terms added to the base tests or with rules for maintaining the rate monotonic assumptions.

Conclusion

With their continuing partnerships with industry companies, SEI and CMU have worked to develop rate monotonic analysis into a widely used and practically applicable technique. By extending an originally restrictive theory, they have succeeded in covering many commonly occurring situations in real-time systems. These efforts have allowed rate monotonic analysis to be used successfully to analyze schedulability in rate monotonic scheduling systems and also in systems that use other scheduling algorithms.

Bibliography

1. Fowler, Priscilla, Levine, Linda. *Technology Transition Push: A Case Study of Rate Monotonic Analysis (Part 1)*, Technical Report CMU/SEI-93-TR-29 ESC-TR-93-203, December 1993.
2. Obenza, Ray, and Mendal, Geoff. *Guaranteeing Real Time Performance Using RMA*, The Embedded Systems Conference, San Jose, CA, 1998.
3. Reeves, Glenn. *What Really Happened On Mars? – Authoritative Account*, research.microsoft.com, 1997.
4. Sha, Lui, Klein, Mark H., and Goodenough, John B. *Rate Monotonic Analysis*, Technical Report CMU/SEI-91-TR-6 ESD-91-TR-6, March 1991.

5. Watson, Ben. *Using PERTS and PERTS*SIM to Analyze End-to-End Completion Times*, Tri-Pacific Software, Alameda CA