Rate Monotonic Analysis in the RMADriver Application

Nate Forman, EE382c Embedded Software Systems

1. Introduction

For a person, consequences for late task completion may include a bad grade or performance review In hard real-time systems such as medical systems, factory control systems, and travel control systems, late task completion can be far more disastrous. Developers of these systems must guarantee that each task meets its deadline.

When a real-time system adheres to rate monotonic conditions, static analysis can be performed to confirm that each task meets its deadline. This paper reviews rate monotonic theory and describes software that has been written to perform rate monotonic schedulability tests.

2. Rate Monotonic Assumptions

Rate monotonic analysis confirms schedulability for tasks in a single-processor real-time system with hard deadlines. It assumes that each task repeats periodically and requires a fixed amount of processor execution time within that period. Figure 1 shows a graphical example of the behavior of two rate monotonic tasks.

Reasoning with rate monotonic analysis requires the system to conform to the following assumptions [4]:

- Task switching is instantaneous
- Tasks account for all processor 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 end of the current period.
- Tasks with shorter periods are assigned higher priorities; no other criteria are considered for priority assignment.
- Task execution is always consistent with rate monotonic priority: a lower priority task never executes when a higher priority task is ready to execute.

The rate monotonic conditions as specified above allow static schedulability tests to be performed on task sets given each period and execution time. Section 3 explains these tests in detail. This set of assumptions is highly restrictive and few real systems, if any, actually conform to all of them. Section 4 explains extensions to the assumptions and adjustments to the tests to include the extensions. Section 5 describes an application that implements the extended rate monotonic analysis tests.

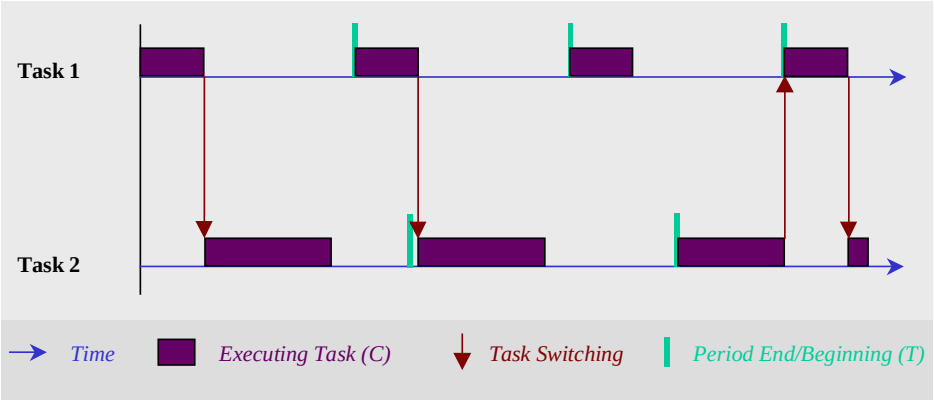

Figure 1: Two tasks executing in a system under rate monotonic conditions.

3. Schedulability Tests

A rate monotonic task, i, has a period length (T_i) and an execution time (C_i). The processor utilization for a set of n tasks is given by the following equation [2]:

$$U_i = \frac{C_i}{T_i}$$

The processor utilization for a set of n tasks is the sum of each individual task's utilization. If the utilization for a set of tasks is over 100%, that set of tasks is clearly unschedulable.

The asymptotic processor utilization bound for a set of n rate monotonic tasks is given by the following equation [2]:

$$U(n) = n\,(2^{\frac{1}{n}} - 1)$$

As the number of tasks approaches infinity, U(n) converges on 69%. Although some runtime schedulers, such as earliest deadline, offer much higher utilization, industry shows a strong preference for the static analysis offered by scheduling under rate monotonic conditions [4].

The **utilization bound test** compares the processor utilization for a set of tasks with the utilization bound for that number of tasks. If the utilization falls below the bound, the tasks will all meet their hard deadlines under rate monotonic conditions. If the utilization is above 100%, the tasks are not schedulable. If the utilization falls between the bound and 100%, the test is inconclusive and a more precise test is required.

For a set of independent periodic tasks, if a task meets its deadline with worst case task phasing, the deadline will always be met [2]. The **response time test** compares the response time of a task with worst-case task phasing to its deadline to see if the task is schedulable. The response time for a task can be calculated as the least fixed-point of the following recurrence:

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

In the recurrence, H represents the subset of the set of tasks whose priorities are higher than the task in question, i. If the task's response time is less than its deadline, the task is schedulable, otherwise it is unschedulable.

4. Extensions

As mentioned above, the rate monotonic assumptions are highly restrictive and few real systems, if any, conform to all of them. The interplay between research and application has resulted in the extension of rate monotonic theory in several ways to make it more broadly applicable. These extensions include:

- Handling for aperiodic tasks
- Preperiod task deadlines
- Nonzero task switching times
- Interrupt handling for top-priority tasks
- Tasks blocking each other because of shared resources

This section focuses on how the schedulability tests are affected by these extensions to rate monotonic theory.

Aperiodic tasks are handled by a process called a sporadic server. Sporadic servers are assigned a period and execution budget like other rate monotonic tasks. These servers operate at their rate monotonic priority and execute aperiodic tasks as needed until their execution budget is depleted. The execution budget is replenished at the beginning of a period after the budget has run out. Because sporadic servers operate under the original rate monotonic assumptions, no adjustments need to be made to the schedulability tests.

For the rest of the above extensions, the response time test does not change drastically. Preperiod task deadlines are handled by comparing the response time to the deadline instead of the end of the period. An extra term for blocking is added into each generation of the recurrence, and twice the task switching delay is added into each task execution time. The extended response time test is as follows:

$$a_0 = B_i + \sum_{j \in H+\{i\}} (C_j + 2S) \qquad a_{n+1} = B_i + C_i + 2S + \sum_{j \in H} \left\lceil \frac{a_n}{T_j} \right\rceil (C_j + 2S)$$

In the above recurrence, S is the amount of task switching time for tasks in the system and B_i is the time that task i spends blocked. For detail on how tasks can block each other without inducing deadlock or unbounded priority inversion, see [4]. For an example of problems caused by unbounded priority inversion, see [3].

These extensions change the utilization bound test much more drastically. First, preperiod deadlines make it necessary to change the utilization bound calculation and perform it for each task separately:

$$U(n, \Delta_i) = \begin{cases} n\left((2\Delta_i)^{1/n} - 1\right) + 1 - \Delta_i, & 0.5 < \Delta_i \leq 1.0 \\ \Delta_i, & \Delta_i \leq 0.5 \end{cases}$$

Δ_i in the above equation is the ratio of the preperiod deadline of task i, D_i, to its period length, T_i.

Interrupt servers, tasks which maintain high priority regardless of period length, cause the utilization for each task to be calculated separately and take into account preemptions from longer period tasks as well as those with shorter periods. The processor utilization for each task can be calculated with the following:

$$f_i = \sum_{j \in Hn} \frac{C_j + 2S}{T_j} + \frac{C_i + 2S}{T_i} + \frac{B_i}{T_i} + \frac{1}{T_i} \sum_{k \in H1} (C_k + 2S)$$

Hn is the set of higher priority tasks that have periods shorter than task i and therefore preempt it more than once. H1 is the set of higher priority tasks that have periods longer than the period of task i, and therefore preempt it only once. Each f_i can be compared to $U(n, \Delta_i)$ as in the utilization bound test before. Any inconclusive results can be tested using the extended response time test. For a detailed explanation of interrupt servers, see [2].

5. Results: The RMADriver Application

Some systems, like the one discussed in [5], provide complete rate monotonic scheduling systems. The solution presented in this paper simply analyzes a task set for schedulability. It does so using the extended schedulability tests explained above.

The *RMADriver* application is a command line application written using IBM VisualAge for Java and is compatible with JDK 1.1.6. It is written to perform the extended rate monotonic schedulability tests on sets of tasks given the necessary

information about those tasks. In addition, the classes used by *RMADriver* have been developed to be reusable as an API for other rate monotonic analysis applications.

The *RMADriver* application presents the user with a command-line, menu-driven interface. This interface allows a user to enter task information or load it from a text file. For convenience, it allows task sets to be saved to text files in the same format that it reads. Tasks in the system consist of a set of information representing execution time, period, deadline, and blocking time as well as whether or not the task is an interrupt server. The application also allows the user to set a task switching delay for the entire system.

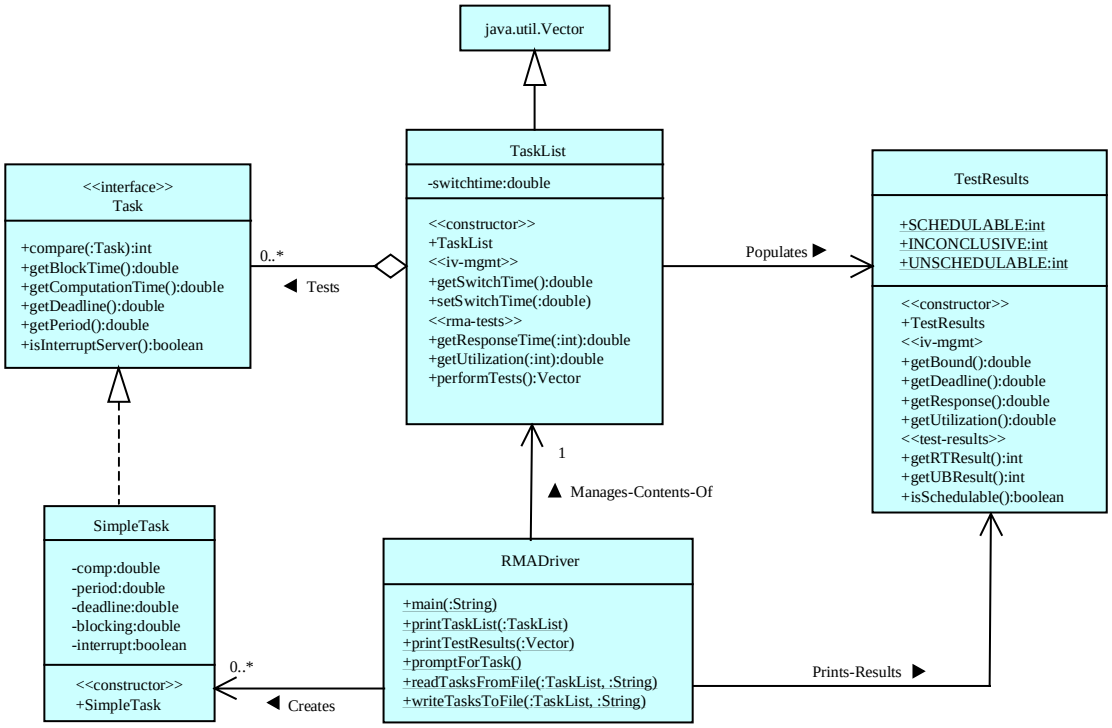

Figure 2: The UML class diagram for the RMADriver application.

When all of the tasks have been entered, the user instructs the application to test the set for schedulability. The application performs the utilization bound test on each

task and reports the results. If the utilization bound test is inconclusive, it also executes the response time test and reports the results.

The advantage to the *RMADriver* application is the design of its *TaskList* component. This Java class is designed to read an interface that returns information about a task rather than depending upon a specific task object. This design allows the *TaskList* to run its tests on any task set that can be adapted to that interface. Therefore, although this specific application only uses *SimpleTask* instances that simply hold the necessary data, the *TaskList* class could be used for rate monotonic analysis on actual real-time systems, as well. In addition, instances of the *TaskList* class report test results in the form of a *Vector* of *TestResults* objects. This form of reporting allows applications to review test results without rerunning the actual tests.

6. Bibliography

1. Fowler, P., and L. Levine, *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, R., and G. Mendal, *Guaranteeing Real-Time Performance Using RMA*, Tutorial 113, The Embedded Systems Conference, San Jose, CA, November 1998.
3. Reeves, G., *What Really Happened On Mars?—Authoritative Account*, research.microsoft.com, 1997.
4. Sha, L., Klein, M. H., and J. B. Goodenough, *Rate Monotonic Analysis*, Technical Report CMU/SEI-91-TR-6 ESD-91-TR-6, March 1991.
5. Watson, B., *Using PERTS and PERTS*SIM to Analyze End-to-End Completion Times*, Tri-Pacific Software, Alameda, CA.