Slack Time Evaluation with RTSJ

Damien Masson and Serge Midonnet
Université Paris-Est
Laboratoire d'informatique de l'institut Gaspard-Monge
UMR 8049 IGM-LabInfo
77454 Marne-la-Vallée Cedex 2, France
{masson, midonnet}@univ-mlv.fr

ABSTRACT

We address in this paper the problem of jointly scheduling hard periodic tasks and soft aperiodic events using the Real-Time Specification for Java (RTSJ). We present the programming constraints of RTSJ and propose slack time evaluation and utilization algorithms which take these constraints into account. We evaluate these algorithms and compare their performances with the Background Scheduling (BS) through simulations.

1. INTRODUCTION

The need for more flexibility in real-time systems leads the community to address the problem of jointly scheduling hard periodic tasks and soft aperiodic events.

The easiest way to achieve this is to schedule all non-periodic tasks at a lower priority (assuming that the tasks are scheduled using a preemptive fixed priority policy). This policy is known as the *Background Servicing (BS)*.

In order to minimize the aperiodic tasks response times whilst guaranteeing the feasibility of the periodic tasks, the periodic task servers were introduced. A server is a periodic task, for which classical response time determination methods are applicable (with or without modifications). The server is in charge of servicing the non-periodic traffic with a limited capacity.

The *Static Slack Stealer*, an algorithm to compute the exact slack available at time t to execute aperiodic tasks at the highest priority, is proposed in [1]. The optimality of this approach for maximizing the aperiodic tasks response times is demonstrated. Unfortunately, the time and memory complexities does not permit to use it.

We propose in this paper to start from RTSJ constraints to set up a slack time approximation algorithm. Then we discuss the slack time utilization issues with RTSJ.

We review in Section 2 the available mechanisms with RTSJ. Section 3 describes our approximate slack evaluation algorithm. Section 4 explicits RTSJ constraints to design slack time evaluation and utilization algorithms. Section 5 presents our simulations methodology and results. We conclude in Section 6.

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

2. APERIODIC TASKS AND RTSJ

The RTSJ proposes two classes `AsyncEvent` and `AsyncEventHandler` to model respectively an asynchronous event and its handler(s). The only way to include an handler in the feasibility process is to treat it as an independent task, and that implies to know at least its worst-case occurring frequency.

The RTSJ does not support any particular task server policy. We propose a solution in [2].

3. AVAILABLE SLACK EVALUATION

Our goal is to construct a slack time evaluation implementable with a minimal RTSJ compliant virtual machine.

The only instants in the system life time where we can easily operate are each periodic task instances begin and end.

We decompose the maximum available slack per task ($S_{i,t}^{max}$) according to Equation 1 where $W_{i,t}^{max}$ is the maximum possible work at priority i regardless of lower priority processes, and $c_i(t)$ is the effective hard real-time work we have to process at the instant t at the priority i.

$$S_{i,t}^{max} = W_{i,t}^{max} - c_i(t) \tag{1}$$

Then the available slack at time t at the highest priority is the minimum of the $S_{i,t}^{max}$.

3.1 Data initialization

Under hypothesis of a synchronous activation of all periodic hard real-time tasks at t_0, we have:

$$\begin{cases} W_{1,t_0}^{max} &= d_1 \\ \forall i, W_{i,t_0}^{max} &= d_i - \sum_{\forall k \in HP(i)} \left\lceil \frac{T_i}{T_k} \right\rceil c_k \\ \forall i, c_i(t_o) &= c_i \end{cases} \tag{2}$$

where c_i, d_i and T_i are respectively the cost, the deadline and the period of the periodic task τ_i. The data initialization has a time complexity in $O(n^2)$, with n the periodic tasks number.

3.2 Dynamic operations

The data are function of time, so they potentially evolve at each clock tick. Between two dates t_1 and t_2, W_i is reduced by $t_2 - t_1$ for any task and $c_i(t)$ is reduced for any task executed during the interval. Eventually, these values can

be increased for the tasks which complete between t_1 and t_2. However, we only need to update them at each periodic task instance begin and end and when an aperiodic event occurs.

1. **End of periodic task τ_i.** Let dt be the elapsed time since the last update of our data. Then, W_i is reduced by the elapsed time since the last update and increased by T_i minus the interference of tasks with higher priority. This interference, given by equation 3, has already be computed during the initialization phase.

$$I_i = \sum_{\forall k \in HP(i)} \left\lceil \frac{T_i}{T_k} \right\rceil c_k \tag{3}$$

I_i is not the real interference at instant t, but the maximal interference. This bound is reach in the worst case scenario: the synchronous activation of all the tasks.

Moreover, for all tasks, W_k is reduced by the elapsed time, and for all tasks with a lower priority than τ_i, W_k is increased by c_i. Finally, $c_i(t)$ is reset to c_i.

$$\begin{cases} \forall j, W_{j,t} & = & W_{j,t'} - dt \\ W_{i,t}^{max} \geq W_{i,t} & = & max(W_{i,t'}, 0) + T_i - I_i \\ \forall j > i, W_{j,t} & = & W_{j,t'} + c_i \\ c_i(t) & = & c_i \end{cases} \tag{4}$$

2. **Beginning of periodic task τ_i.**

The maximum available process time is reduced by dt at all priority level. Let j be the previous activity priority level. If j is in the hard real-time priority range (noted $hrtp$), then $c_j(t)$ is reduced by dt.

$$\begin{cases} \forall k, W_{k,t} & = & W_{k,t'} - dt \\ if\ j \in hrtp, c_j(t) & = & c_j(t') - dt \\ if\ W_{i,t} < c_i(t), W_{i,t} & = & c_i(t) \end{cases} \tag{5}$$

3. **Soft real-time aperiodic task τ_α arrival.**

As for the previous cases, the maximum available process time is reduced by dt at all priority level as W_j if the previous activity priority, j, is in the hard real-time priority range. Then, the available slack at the highest priority is computed.

$$\begin{cases} \forall k, W_{k,t} & = & W_{k,t'} - dt \\ if\ j \in hrtp, c_j(t) & = & c_j(t') - dt \\ S_{i,t}^{max} \geq S_{i,t} & = & \min\limits_{i \in hrpr} W_{i,t} - c_i(t) \end{cases} \tag{6}$$

4. RTSJ CONSTRAINTS

4.1 Slack approximation

In order to act at the beginning and at the end of the periodic tasks instances, we overload the `waitForNextPeriod()` method of `RealTimeThread`. This method is called at the end of each periodic job and is blocking until the next job execution.

The slack stealer may be implemented as an `AsyncEventHandler` bound to a special `AsyncEvent` and with the maximal available priority.

The soft aperiodic events can be implemented with several `AsyncEvent` all bound to the `AsyncEventHandler` representing the slack stealer. In this way, we can call the slack stealer every time a soft aperiodic event is released and before and after each instance of hard periodic tasks.

4.2 Slack Utilization

With Java, it is impossible to resume a previously stopped thread. The RTSJ offers a mechanism to asynchronously interrupt a RT-thread, but it does not allow to resume it later. Moreover, in most implementation of RTJVM, the dynamic change of the priority is not supported.

Due to these limitations, we cannot begin the execution of a soft task at the highest priority if there is not enough slack to permit its completion.

When there is available slack, we schedule the execution of the first soft pending task with a cost lower or equal than the available slack at the highest priority. When there is no slack and no pending periodic task, we schedule in BS the first pending aperiodic task.

5. SIMULATIONS

We simulate the same real-time systems with the three following aperiodic policies: a BS, a slack stealing with an exact computation of the slack and one with our approximation of the slack.

5.1 Methodology

We measure the mean response time of soft tasks with different aperiodic and periodic loads.

First, we generate groups of ten periodic task sets with utilization levels of 30, 50, 70 and 90%. The results presented in this section are based on averages over a group of ten task sets. Then, we generate groups of ten aperiodic task sets with a range of utilization levels (plotted on the x-axis in the following graph). Our simulations end when all soft tasks have been served.

We experiment on a GPL home made event-based simulator.

5.2 Results

Our simulations show an improvement over a simple BS. However, we are nearer from a BS than from the same policy using an exact computation of the slack. Moreover, the higher the periodic charge is, the worst the results are.

6. CONCLUSIONS

In this paper, we address the problem of jointly scheduling hard periodic tasks and soft aperiodic events with RTSJ. We present the programming constraints of RTSJ and propose a simple slack time evaluation algorithm. We describe a possible implementation of this algorithm. We describe a RTSJ compliant slack utilization algorithm. We simulate this algorithm with an exact slack computation and with our approximation. If we obtain better results than a simple BS, these simulations point out that our approximation of the available slack is too large.

7. REFERENCES

[1] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks fixed priority preemptive systems. In *proceedings of the 13th IEEE Real-Time Systems Symposium*, pages 110–123, Phoenix, Arizona, December 1992.

[2] D. Masson and S. Midonnet. The design and implementation of real-time event-based applications with RTSJ. In *WPDRTS (in proceedings of IPDPS)*, page 148, Long Beach, CA USA, March 2007.