Guaranteeing Schedulability with Server Mechanisms for RTOS Overhead

Kiyofumi Tanaka
School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
kiyofumi@jaist.ac.jp

Kazuki Hasegawa
School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
kazuki_h@jaist.ac.jp

ABSTRACT

We propose techniques for guaranteeing schedulability including RTOS runtime overhead. In the techniques, we introduce two servers with planned capacity: periodic management server for invoking periodic tasks and aperiodic management server for aperiodic events such as task completion. Then, we provide the schedulability test considering the servers' utilization. In addition, we show techniques for slack reclaiming to improve the aperiodic tasks' responsiveness by utilizing capacity (or slack) of the servers which is left unused. The evaluation show that average response times for aperiodic tasks can be improved by up to 20.7%.

1. INTRODUCTION

Especially in hard real-time systems, it is important to guarantee the schedulability of all the hard tasks. As a representative method of assuring schedulability, one which takes processor utilization by tasks into account has been established, where the sum of the tasks' utilization is kept lower than the least upper bound of the task set (U_{lub})[1]. However, this is not sufficient since the RTOS overhead can make the system unschedulable.

It is possible to take estimated RTOS overhead into account for schedulability analysis [1, 2]. However, the estimation about the frequency of preemption or context switching can be done only for periodic tasks. When aperiodic tasks exist, it is not possible to assure that actual RTOS overhead is within the estimated amount. In addition, the analysis model depends on an ideal assumption that every single event can occur only at a multiple of the system tick. It cannot be expected that aperiodic events and a system tick synchronize since the system tick is relatively long.

We propose techniques for guaranteeing schedulability including RTOS overhead, where we introduce two servers, periodic management and aperiodic management servers, dedicated to RTOS processing and guarantee the system schedulability. In addition, we improve the aperiodic tasks' responsiveness by utilizing unused capacity of the servers.

2. SERVERS FOR RTOS PROCESSING

In the system model, there are two tick units: system tick and processing tick. The former is units of time to express time at which periodic tasks can be requested to run and time for which tasks are supposed to run, that is, worst-case execution time (WCET). The latter is units of time to express the other time properties; tasks' actual execution time, actual finishing time, and RTOS's processing time.

Two servers for RTOS are deployed. One is a periodic management server which takes charge of invocation processing of tasks and, if necessary, task switching. The other is an aperiodic management server which processes aperiodic events such as tasks' completion and task switching. These two servers have the shortest period and the highest-priority in the RM context since the RTOS processing should be performed in priority to other application tasks' processing.

2.1 Periodic Management Server

Periodic tasks are supposed to be requested in synchronization with system ticks. For example, a periodic task with period of four system ticks is requested every four system ticks. When this task is included in the system, the periodic management server processes the invocation every four system ticks. On each system tick timing, there can be two or more invocations of tasks. The periodic management server exploits the Polling Server algorithm [3] and is given capacity every system tick. The capacity should be large enough to process the maximum number of invocations that can occur on system tick timing and it can be estimated in the hyperperiod of the task set. Every system tick timing, the server processes the task invocations and task switching if necessary. When fewer or no tasks requested and the capacity is left, the capacity is abandoned.

2.2 Aperiodic Management Server

The second server, aperiodic management server, manages aperiodic events such as tasks' completion including task switching. Tasks can finish at any timing different from system ticks. Therefore, we use Deferrable Server [3] which preserves the capacity during the system tick period.

In the task model in this study, WCETs are regarded as supposed execution times and the WCETs are estimated as the integral multiple of a system tick. We try to guarantee the schedulability assuming the WCETs. From this assumption, at most one task would finish in a system tick. This leads to that one processing tick for the server capacity is enough to guarantee the schedulability. If a task is going to finish before it has spent its WCET, the capacity may be already exhausted. In this case, blocking the system until the next system tick timing does not influence the schedulability. However, this is a waste of the system resources. Therefore, it is effective to provide two or more processing ticks as the capacity to improve the system utilization.

EWiLi'15, October 8th, 2015, Amsterdam, The Netherlands.

2.3 Merging Two Servers

Figure 1 shows a schedule example of a system including both the periodic and aperiodic management servers. The periodic server is scheduled whenever periodic tasks are released, while the aperiodic management server is scheduled immediately after each periodic instance finishes.

Guaranteeing the schedulability with the periodic and aperiodic management servers requires the following condition to be met. This is derived from the scheduability condition for the Deferrable Server. Developers can statically check schedulability of the systems by using this formula.

$$\sum_{i=1}^{n} C_i/T_i + C_{PMS}/T_{PMS} \leq (n+1)\left[\left(\frac{C_{AMS}/T_{AMS}+2}{2C_{AMS}/T_{AMS}+1}\right)^{1/(n+1)} - 1\right] \quad (1)$$

3. SLACK RECLAIMING

The RTOS servers proposed in the previous section are apt to generate slack which is capacity being left unused due to infrequent RTOS processing. For example, in Figure 1, the total capacity provided to the two servers is 30 between $t = 0$ and $t = 10$. However, the amount actually consumed is 8, which is less than 30% of the total. By giving the unused capacity to non-real-time aperiodic tasks, the response times of the aperiodic tasks can be shortened since the capacity of the servers has the highest priority. The amount of capacity which can be given to aperiodic tasks can be decided by considering periodicity of periodic tasks.

4. EVALUATION

The evaluation targets synthetic task sets that are generated using probabilistic distribution.We prepared 100 periodic task sets for each of utilization (U_p) between 50% and 70%. In addition, we prepared 10 aperiodic task sets that follow Poison arrival and exponential service times. The utilization by the aperiodic tasks is about 1%. The periodic and aperiodic management servers have the same units for capacity as the number of periodic tasks in each task set.

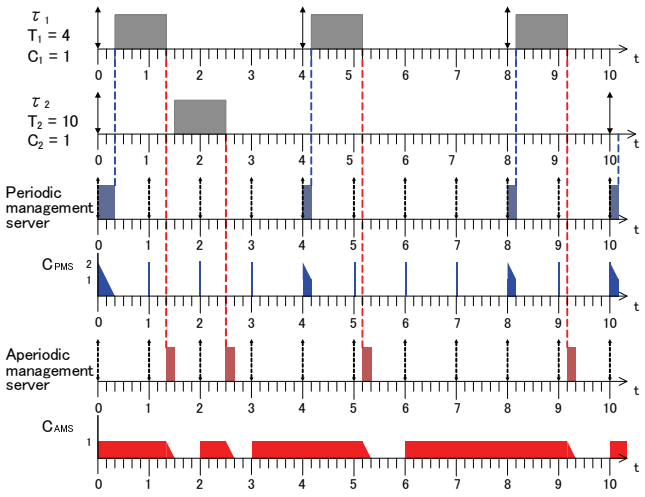

Figure 1: System with periodic and aperiodic management servers.

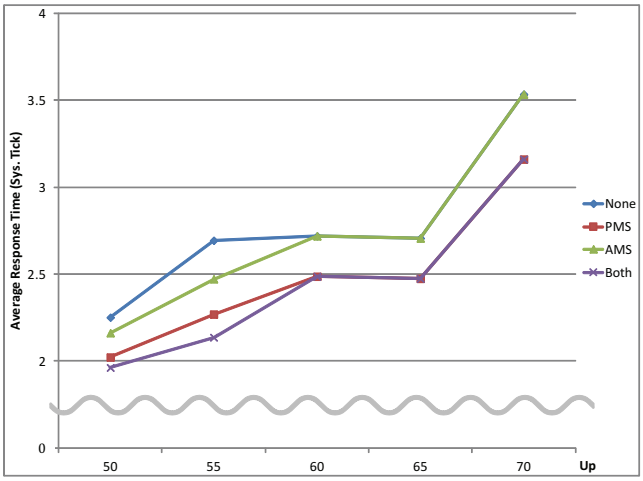

Figure 2: Average response time of aperiodic tasks (1 system tick = 50 processing ticks).

Figure 2 shows average response times of aperiodic tasks when a processing tick equals to 1/50 of a system tick. "None" corresponds to the results without slack reclaiming. "PMS" and "AMS" use slack reclaiming by the periodic management server and the aperiodic management server, respectively. "Both" means slack reclaiming by both the servers. The improvement by the resource reclaiming becomes up to 20.7%.

5. CONCLUSIONS

In this paper, we proposed techniques to assure schedulability including RTOS processing overhead. The proposed techniques include two high-priority servers, a polling server for periodic management and a deferrable server for aperiodic management, which are in charge of RTOS processing. It is possible to guarantee schedulability by performing schedulability test in Section 2.3 off-line.

The proposed servers tend to have unused capacity in system periods when periodic tasks do not arrive or finish. To avoid wasting the unused capacity, we proposed slack reclaiming techniques for aperiodic responsiveness. In the evaluation, the effects are up to 20.7%.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 15K00073.

6. REFERENCES

[1] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, 3rd ed.* Springer, Reading, Massachusetts, 2011.

[2] K. Jeffay and D. L. Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Proc. of IEEE Real-Time Systems Symposium*, pages 212–221, December 1993.

[3] J. P. Lehoczky, L. Sha, and J. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proc. of IEEE Real-Time Systems Symposium*, pages 261–270, December 1987.