Scheduling Imprecise Computations with Wind-up Parts

Hidenori Kobayashi
School of Science for
Open and Environmental Systems
Keio University
Yokohama, Kanagawa,
223-8522, Japan
kobahide@ny.ics.keio.ac.jp

Nobuyuki Yamasaki
Dept. of Information
and Computer Science
Keio University
Yokohama, Kanagawa,
223-8522, Japan
yamasaki@ics.keio.ac.jp

Yuichiro Anzai
Dept. of Information
and Computer Science
Keio University
Yokohama, Kanagawa,
223-8522, Japan
anzai@ics.keio.ac.jp

Abstract

The imprecise computation model provides the ability to cope with unpredictable workloads. However, there is no consistent way on how to terminate the computation in its early stage. This paper describes a novel approach for safely terminating imprecise computations. First, a new logical part called *wind-up part* is added to the imprecise computation model. This wind-up part is used by application programmers to explicitly specify any operations required to be performed before its optional part is terminated. We have also developed an algorithm based on the mandatory-first earliest deadline first strategy to schedule computations based on the proposed model.

1 Introduction

Real-time systems used to be constructed on simple hardware and often contained one simple application whose behavior had been well analyzed. This lead to the development of the worst case model still used these days. However, this worst case model is no longer effective in modern real-time systems, because a tight upper bound on the execution time is now harder to compute. This is due to complex hardware, multiprogramming, and the move towards the open real-time system. A looser upper bound on the worst case execution time leads to inefficient use of resource, which is not tolerable in systems where cost is an important factor. It seems that it is high time we shifted from the worst case model to a more appropriate computation model.

The new computation model requires the ability to provide flexibility as a whole system to handle the uncertainty in workloads so that the average resource utilization ratio is kept high while no hard deadlines are missed. The imprecise computation model presented in [1] meets such demands. Tasks based on the imprecise computation model consists of two parts. The mandatory part produces logically correct result whereas the optional part merely enhances the quality of the result. During overloads, optional parts are terminated or discarded to adjust the system workload.

To our knowledge, no work on the imprecise computation model is efficient and powerful enough to be adopted in distributed real-time systems whose environment is highly unpredictable. For example, in [2], Hansson et al. describes an approach using OR-ULD for resolving transient overloads, but it assumes that the worst case execution time for every tasks is known a priori. In [3], Hull et al. describes ICS, an environment for developing imprecise systems. However, it is not clear how the method can be applied to distributed systems.

In this paper, we first point out that unrealistic assumptions made in previous imprecise systems were accounted for by a mismatch between the imprecise computation model and real-world applications. We then, focusing on the way computations are terminated, present a practical approach for handling uncertainty in workloads.

2 The Imprecise Computation with Wind-up Part Model

One of the problems that the application programmers encounter on using the imprecise computation model is that there is no way to specify how the optional part should be terminated. This requires that each application be constructed to tolerate sudden termination. However, real-world applications often require some compensation codes for the early termination. Examples of such activities include preparing for the next period if the task is periodic, transmitting

the result to other nodes, and storing the quality of the result and other information for statistical feedbacks. These operations are all mandatory in the sense that they can not be discarded or terminated. Moreover, it must not be executed before the optional part.

Since the imprecise computation model does not allow any mandatory portion to follow its optional portion, the only choice left to the application programmer is to totally rely on the support provided in the run-time system. Consequently, the constructed application is very dependent on the system software. Since the types of support given by operating systems differ, it can not be ported to another systems without modifying its structure.

Our solution to these problems is to introduce another mandatory part called *wind-up part*. This part is always executed after the optional part, whether the computation is terminated, discarded, or completed. This is because the operations needed to terminate the computation are usually also needed for normal completion. The advantage of this extension is that it allows the programmer to explicitly specify exact operations needed to terminate the application. As a result, it is no longer fully dependent on the support provided by the system software.

In the following, we define a task τ_i based on this model with following parameters given a priori.

r_i**:** The absolute release time of the task.

m_i**:** The worst case execution time of the mandatory part. This does not include the value of w_i.

w_i**:** The worst case execution time of the wind-up part.

d_i**:** The deadline of the task relative to r_i.

T_i**:** The period of the task.

The execution time of the optional part is intentionally omitted from the above parameters because its variations tend to be very wide. This is because the behavior of the optional part is highly input driven. For instance, in a target tracking application, the optional part may corresponds to analyzing as many targets as possible, while the mandatory part corresponds to analyzing just one target which is the nearest to itself.

3 Mandatory-First with Wind-up Part Algorithm

The drawback of using the wind-up part is that it complicates the scheduling algorithm. The algorithm for scheduling tasks based on the imprecise computation with wind-up part model is required to guarantee that any wind-up part completes before its deadline. It should also ensure that no wind-up part is executed before the optional part of the same task, as well as scheduling the mandatory part before the optional part. Additionally, the wind-up part should be executed in a preemptive context because its execution time can be arbitrarily long.

We have developed an algorithm called *Mandatory-First with Wind-up Part (M-FWP)* which fulfills all the requirements. It assumes that the system consists of periodic tasks with hard deadline. Their deadlines are assumed to be same as their period. It also assumes that periodic tasks are not activated and suspended dynamically.

The entire algorithm is shown in Fig. 1. It extends the mandatory-first strategy addressed in [4]. In order to meet the demands, it manages two queues in an EDF manner. One is called the mandatory ready queue (MQ) and holds tasks that are ready to execute the mandatory part or the wind-up part. The other is called the optional ready queue (OQ) and holds those ready to execute its optional part. A ready task is always sought first in the MQ and then in the OQ, which forces every mandatory or wind-up part to have a higher priority than any optional part.

The most important point in scheduling wind-up parts is the management of processor time that can be allocated to optional parts. For this purpose, the M-FWP algorithm holds a variable $R_i(t)$ called *remaining* for every ready task. The value of $R_i(t)$ represents the maximum amount of time that is surely available to one of the parts which is executed by task τ_i at time t. The remaining should be set, in the beginning of each part, to its worst case execution time and decreased as the execution proceeds. However, the worst case execution time of the optional part can not be used here because it is neither known nor guaranteed. Instead, the value of remaining for the optional part of task τ_i is set to the following:

$$\begin{aligned} o_i(t) \;=\; & d_i - t - w_i - \sum_{\tau_m \in \Gamma_m} R_m(t) \\ & - \sum_{\tau_o \in \Gamma_o(\tau_i)} (R_o(t) + w_o) \\ & - \sum_{\tau_{op} \in \Gamma_{op}(\tau_i)} D_{op}(r_{op} + T_{op},\ d_i - w_i) \\ & - \sum_{\tau_s \in \Gamma_s} D_s(r_s,\ d_i - w_i) \end{aligned} \tag{1}$$

1. If one of the following event occurred:
   (a) a task τ_i has become ready:
      i. initialize its remaining to m_i; and
      ii. enqueue τ_i to MQ.
   (b) a task τ_i has completed its mandatory part:
      i. set its remaining to $\max(o_i(t), 0)$;
      ii. move τ_i from the MQ to the OQ; and
      iii. subtract total of $R_i(t)$ from the remaining of the other tasks in the OQ with a later deadline.
   (c) a task τ_i at the head of the OQ has completed its optional part or $R_i(t)$ has become zero:
      i. add $R_i(t)$ to the remaining of the next ready task in the OQ if it exists;
      ii. set $R_i(t)$ to w_i; and
      iii. move τ_i from the OQ to the MQ.
   (d) a task has completed its wind-up part:
      i. dequeue the task from the MQ.
2. If a ready task exists, execute the task and decrease its remaining. A ready task is first searched in the MQ and then in the OQ.
3. Otherwise, do nothing.

Figure 1: The M-FWP Algorithm

where

$$D_i(t_1, t_2) = \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor (m_i + w_i) + \min\left(m_i + w_i,\ (t_2 - t_1) \bmod T_i\right),$$

Γ_m**:** tasks in the MQ,

$\Gamma_o(\tau_i)$**:** tasks in the OQ whose deadline is earlier than or equal to that of τ_i,

$\Gamma_{op}(\tau_i)$**:** periodic tasks which belong to $\Gamma_o(\tau_i)$, and

Γ_s**:** periodic tasks which are not ready.

Here, $D_i(t_1,\ t_2)$ gives an upper bound for the processing time demanded by task τ_i between t_1 and t_2, where t_1 is the beginning of its period. Consequently, $o_i(t)$ gives a lower bound for the amount of time not allocated to any mandatory or wind-up part before d_i.

	r_i	m_i	w_i	T_i
T_1	0	1	1	9
T_2	0	1	1	5

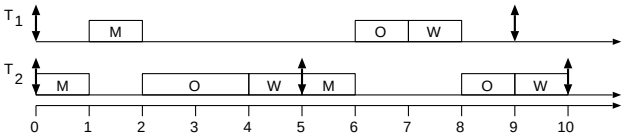

Figure 2: Sample Schedule

Since the last wind-up part included in the second term may not have to be executed before d_i in practice, the value given by $D_i(t_1, t_2)$ is merely an upper bound. This situation is illustrated in Fig. 2 where two periodic tasks T_1 and T_2 are scheduled according to the M-FWP algorithm. The execution of the mandatory, optional, and wind-up parts are shown as boxes which contain letter M, O, and W, respectively. The value of $o_1(2)$ is calculated as 1 according to Eq. (1) despite the fact that it is actually 2, because its deadline is earlier than that of T_2 in the second cycle. Thus, the value given by $o_i(t)$ can be negative even if the system is schedulable.

The condition which the M-FWP algorithm requires to guarantee $R_i(t) \geq 0$ for all tasks in the system is that the essential system utilization defined as:

$$U = \sum_{\tau_i} \frac{m_i + w_i}{T_i} \tag{2}$$

does not exceed one. Under this condition, we can safely set the remaining to zero even if the value calculated by $o_i(t)$ is negative due to the error contained in $D_(t_1, t_2)$.

When a new remaining for τ_i is set on the start of its optional part, the remaining of tasks in the OQ whose deadline is later than d_i may decrease. This is because the time requested as $o_i(t)$ is obtained from above tasks by collecting, in the increasing order of the deadline, as much time as possible until enough time is obtained or no more task exists in the OQ.

4 Experimental Results

We have developed a real-time operating system called *RT-Frontier* which implements the proposed scheme on a network of *Responsive Processors*, each of which is connected by *Responsive Links*. Details of the hardware are found in [5].

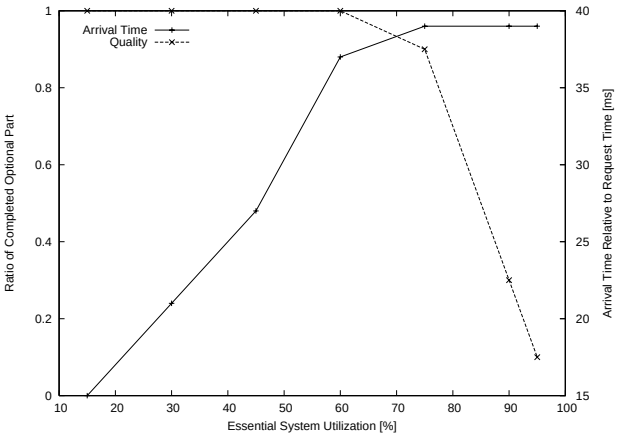

Figure 3: Quality and Arrival Time of Results

We set up an client-server application in order to evaluate the ability of the proposed scheduling algorithm to ensure the completions of wind-up parts. In the experiment, a client was set up to wake up every 40 ms to make a request and, at the same time, check the result of the last request. A server was activated by the request and returns, in its wind-up part, the ratio of the completed optional part, which corresponds to the quality of the result in real applications. The execution time of the mandatory, optional and wind-up part of the server were set to $4ms$, $10ms$, and $1ms$, respectively, though the execution time of the optional part was hidden from the scheduler. The relative deadline of the server was set to $38ms$, considering the communication latency.

The result is shown in Fig. 3. The normalized average quality of the results obtained by the client task is shown with a dotted line which uses the vertical axis on the left side. It shows that the optional parts of the server are gracefully cut to adjust the system utilization and that without the wind-up part, no result was transmitted. The solid line and the vertical axis on the right side show the average arrival time of the result on the client node. It shows that each results successfully arrived to the client node before the deadline.

5 Conclusions

We have presented a new form of the imprecise computation model for handling unpredictable workloads in real-time systems. We have also developed a scheduling algorithm for such computations. The proposed approach makes it easier to realize flexible computations in real-world applications in two points: no particular knowledge of optional parts is required; and the programmer can explicitly specify what to be done on its premature completion, for example, transmitting the results. The presented approach also allows the application to be reused in different systems, because the effect of changes in its environment is safely absorbed by the optional part as long as the mandatory part and the wind-up part are schedulable. As future work, we are currently working to extend the approach to support 0/1-constraint optional parts.

Acknowledgement

This study was performed through Special Coordination Funds of the Ministry of Education, Culture, Sports, Science and Technology of the Japanese Government.

References

[1] K. Lin, S. Natarajan, and J.-S. Liu, "Imprecise Results: Utilizing Partial Computations in Real-Time Systems," in *Proceedings of the IEEE 8th Real-Time Systems Symposium*, pp. 210–217, December 1987.

[2] J. Hansson, M. Thuresson, and S. H. Son, "Imprecise Task Scheduling and Overload Management using OR-ULD," in *Proceedings of the Seventh International Conference on Real-Time Computing Systems and Applications*, pp. 307–314, December 2000.

[3] D. Hull, W. Feng, and J.-S. Liu, "Enhancing the Performance and Dependability of Real-Time Systems," in *Proceedings of the IEEE International Computer Performance and Dependabilility Symposium*, pp. 174–182, April 1995.

[4] H. Aydin, P. Mejia-Alvarez, R. Melhem, and D. Mossé, "Optimal Reward-Based Scheduling of Periodic Real-Time Tasks," in *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pp. 79–89, December 1999.

[5] N. Yamasaki, "Design and Implementation of Responsive Processor for Parallel/Distributed Control and Its Development Environment," *Journal of Robotics and Mechatronics*, vol. 13, no. 2, pp. 125–133, 2001.