Semi-Fixed-Priority Scheduling with Multiple Mandatory Parts*

Hiroyuki Chishiro[†‡] and Nobuyuki Yamasaki[‡]

[†]Department of Computer Science, The University of North Carolina at Chapel Hill, NC, USA
[‡]Department of Information and Computer Science, Keio University, Yokohama, Japan
chishiro@cs.unc.edu,yamasaki@ny.ics.keio.ac.jp

Abstract

An imprecise computation model has the advantage of supporting overloaded conditions in dynamic real-time environments, compared to Liu and Layland's model. However, the imprecise computation model is not practical because the termination of each optional part cannot guarantee the schedulability. In order to guarantee the schedulability of the termination of the optional part, a practical imprecise computation model is presented. In the practical imprecise computation model, each task has multiple mandatory parts and optional parts to support many imprecise real-time applications. The practical imprecise computation model is supported by dynamic-priority scheduling on uniprocessors. Unfortunately, dynamic-priority scheduling is difficult to support multiprocessors. In contrast, semi-fixed-priority scheduling, which is part-level fixed-priority scheduling, supports only two mandatory parts so that supported imprecise real-time applications are restricted. This paper presents semi-fixed-priority scheduling with multiple mandatory parts on uniprocessors and multiprocessors respectively. In addition, this paper explains how to calculate the optional deadline of each task, which is the termination time of optional part. The schedulability analysis shows that semi-fixed-priority scheduling strictly dominates fixed-priority scheduling. Thanks to semi-fixed-priority scheduling with multiple mandatory parts, many imprecise real-time applications can be supported.

1. Introduction

Nowadays real-time applications have been encountering overloaded conditions in dynamic environments. For example, autonomous mobile robots such as Roomba [5] have been used in such dynamic environments. In order to detect and to avoid obstacles in such dynamic environments, real-time scheduling must support overloaded conditions. However, traditional real-time scheduling is usually based on Liu and Layland's model [10], which does not support overloaded conditions. In addition, multiprocessors can overcome such overloaded conditions by increasing processor capacities. In order to perform real-time scheduling in such overloaded conditions, an imprecise computation model [9], which improves the quality of result with timing constraints, is presented.

The imprecise computation model has the advantage of supporting overloaded conditions in dynamic real-time environments, compared to Liu and Layland's model. The crucial point in the imprecise computation model is that the computation is split into two parts: mandatory part and optional part. A mandatory part as a real-time part affects the correctness of the result and an optional part as a non-real-time part only affects the quality of the result. By restricting the execution of the optional part only after the completion of the mandatory part, imprecise real-time applications can provide the correct output with lower quality, by terminating the optional part. In overloaded conditions, the imprecise task terminates its optional part and generates the result with lower quality. However, an imprecise computation model is not practical because the termination of each optional part cannot guarantee the schedulability. In order to guarantee the schedulability of the termination of the optional part, a practical imprecise computation model [7] is presented.

In the practical imprecise computation model, each task has multiple mandatory parts and optional parts to support many imprecise real-time applications. The practical imprecise computation model is supported by dynamic-priority scheduling on uniprocessors [7]. Unfortunately, dynamic-priority scheduling is difficult to support multiprocessors. In contrast, semi-fixed-priority scheduling [2], which is part-level fixed-priority scheduling, supports only two mandatory parts so that supported imprecise real-time

*This research was supported in part by CREST, JST. This research was also supported by Grant in Aid for the Global Center of Excellence Program for "Center for Education and Research of Symbiotic, Safe and Secure System Design" from the Ministry of Education, Culture, Sport, and Technology in Japan. In addition, this research was also supported in part by Grant in Aid for the JSPS Fellows.

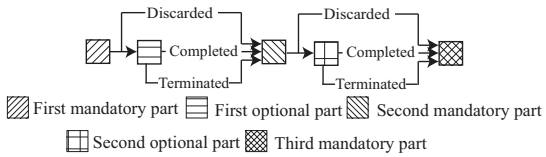

Figure 1. Practical imprecise task with three mandaotry parts and two optional parts

applications are restricted.

This paper presents semi-fixed-priority scheduling with multiple mandatory parts on uniprocessors and multiprocessors respectively. In addition, this paper explains how to calculate the optional deadline of each task, which is the termination time of optional part. The schedulability analysis shows that semi-fixed-priority scheduling strictly dominates fixed-priority scheduling. That is to say, one task set is schedulable by semi-fixed-priority scheduling if the task set is schedulable by fixed-priority scheduling.

The contribution of this paper is to support semi-fixed-priority scheduling with multiple mandatory parts on uniprocessors and multiprocessors. Thanks to multiple mandatory parts, many imprecise real-time applications can be supported. We believe that semi-fixed-priority scheduling in the practical imprecise computation model is widely used in dynamic environments.

The remainder of this paper is organized as follows. Section 2 describes system model. Section 3 introduces semi-fixed-priority scheduling with two mandatory parts. Section 4 presents semi-fixed-priority scheduling with multiple mandatory parts. We compare our work with related work in Section 5. Finally we offer concluding remarks in Section 6.

2. System Model

2.1 Practical Imprecise Computation Model

The practical imprecise computation model [7] has multiple mandatory parts and optional parts. Thanks to the following mandatory parts after executing optional parts, each practical imprecise task guarantees the schedulability of the processing to output the result. Figure 1 shows the practical imprecise task with three mandatory parts and two optional parts. Each practical imprecise task has three execution paths for optional part: discarded, completed and terminated. Therefore, each practical imprecise task can output the proper quality of result without deadline miss due to the overrun of the optional part.

One of the primary target applications in the practical imprecise computation model is a lip reading task by a sensor fusion of a camera and a microphone [13]. The lip reading task has three mandatory parts and two optional parts. Each part in the practical imprecise task executes the following operation.

- First mandatory part: inputs the visual information from a camera. There are some metrics to read the lip: the width of the lip, the height of the lip and the variation of the height of the lip. In the first mandatory part, one of the metrics is executed to generate the result with low quality.
- First optional part: performs other metrics except the metric performed in the first mandatory part. More metrics are performed in the first optional part, the result from the camera has higher quality.
- Second mandatory part: inputs the auditory information from a microphone. There are also some metrics to analyze what a person speaks: log power spectrum and LPC-derived mel-cepstrum. In the second mandatory part, one of the metrics is also executed to generate the result with low quality.
- Second optional part: also performs other metrics except the metric performed in the second mandatory part. More metrics are performed in the second optional part, the result from the microphone has higher quality.
- Third mandatory part: performs the fusion of the visual and auditory information by neural network [8] or hidden markov model [11].

This paper assumes that the system has M identical processors and a task set Γ consisted of n periodic tasks with implicit deadlines. Task τ_i is represented as the following tuple $(T_i, D_i, OD_i, m_i, o_i)$: where T_i is the period, D_i is the relative deadline, OD_i is the group of the relative optional deadline, m_i is the total Worst Case Execution Time (WCET) of the mandatory parts and o_i is the total Required Execution Time (RET) of the optional parts. The total WCET of mandatory parts of task τ_i is $m_i = \sum_{l=1}^{n_i^m} m_i^l$, where n_i^m is the number of mandatory parts and m_i^l is the WCET of the l^{th} mandatory part. On the other hand, the total RET of optional parts is $o_i = \sum_{l=1}^{n_i^o} o_i^l$, where n_i^o is the number of optional parts and o_i^l is the RET of the l^{th} mandatory part. The group of the relative optional deadline of task τ_i is $OD_i = \{OD_i^1, OD_i^2, ..., OD_i^{n_i^o}\}$, where OD_i^l is the l^{th} relative optional deadline of task τ_i. Since the optional deadline is a time to terminate an optional part, the number of optional deadlines is equal to that of optional parts n_i^o. The detail of the optional deadline is shown in Subsection 2.2. The relative deadline D_i of each task τ_i is equal to its period T_i. The j^{th} instance of task τ_i is called job

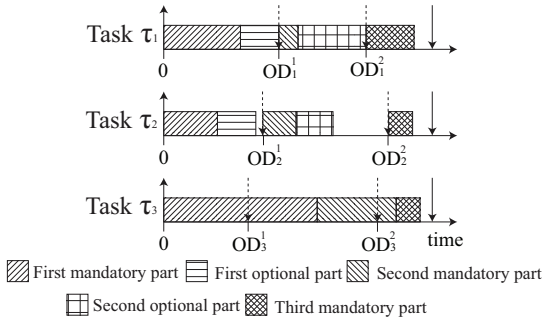

Figure 2. Optional deadline

$\tau_{i,j}$. The utilization of each task is defined as $U_i = m_i/T_i$. The reason why U_i does not include o_i is because the optional part of task τ_i is a non-real-time part so that completing it is not relevant to scheduling the task set successfully. Hence, the system utilization within n tasks can be defined as $U = \sum_i U_i/M$. All tasks are ordered by increasing their periods and task τ_1 has the shortest period in the task set.

2.2 Optional Deadline

An optional deadline is defined as a time when an optional part is terminated and a following mandatory part is released. Each following mandatory part is ready to be executed after each optional deadline and can be completed if each mandatory part is completed by each optional deadline. Each optional deadline is set to the time as late as possible to expand the executable range of each optional part. The following mandatory part must not miss its deadline if there is idle processor time or lower priority tasks are executed between the time when the mandatory part is completed and the following mandatory part is released. If each task does not complete its mandatory part until its corresponding optional deadline, the task may miss its deadline. Thanks to this definition, semi-fixed-priority scheduling does not degrade the schedulability compared to fixed-priority scheduling [2, 3].

Figure 2 shows the behavior of optional deadlines. Solid up arrow, solid down arrow and dotted down arrow represent release time, deadline and optional deadline respectively. Now we describe the execution of each task.

- Task τ_1 executes the first mandatory part. When task τ_1 completes the first mandatory part, task τ_1 executes the first optional part. When OD_1^1 expires, task τ_1 terminates the first optional part and executes the second mandatory part. After completing the second mandatory part, task τ_1 executes the second optional part. When OD_1^2 expires, task τ_1 executes the third mandatory part.

- Task τ_2 executes the first mandatory part and then the first optional part. When task τ_2 completes the first optional part, OD_2^1 does not expire so that task τ_2 sleeps until OD_2^1. Next task τ_2 executes the second mandatory part and the second optional part and sleeps until OD_2^2. After OD_2^2, task τ_2 executes the third mandatory part.

- Task τ_3 executes the first, the second and the third mandatory parts continuously. Because task τ_3 does not complete the first mandatory part until OD_3^1 and the second mandatory part until OD_3^2.

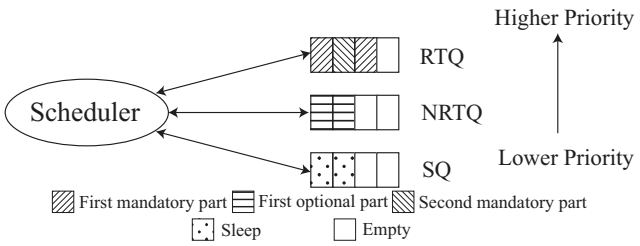

Figure 3. Task queue

3. Semi-Fixed-Priority Scheduling with Two Mandatory Parts

Semi-fixed-priority scheduling [2] is defined as part-level fixed-priority scheduling. That is to say, semi-fixed-priority scheduling fixes the priority of each part in the practical imprecise task and changes the priority of each practical imprecise task only in the two cases: (i) when the task completes its mandatory part and executes its following optional part; (ii) when the task terminates or completes its optional part and executes the following mandatory part.

Rate Monotonic with Wind-up Part (RMWP) [2] is a semi-fixed-priority scheduling algorithm on uniprocessors. As shown in Figure 3, RMWP manages three task queues: Real-Time Queue (RTQ), Non-Real-Time Queue (NRTQ) and Sleep Queue (SQ). In this case, each task has two mandatory parts and one optional part. The RTQ holds tasks, which are ready to execute their mandatory parts in Rate Monotonic (RM) order [10]. One task is not allowed to execute their mandatory parts simultaneously. The NRTQ holds tasks, which are ready to execute their optional parts in RM order. Every task in the RTQ has higher priority than that in the NRTQ. The SQ holds tasks, which complete their optional parts by their optional deadlines or their last mandatory parts by their deadlines.

The optional deadline of each task in RMWP with two mandatory parts on uniprocessors is calculated by the following theorem.

Theorem 1 (From literature [2])**.** *The first relative optional deadline OD_k^1 of task τ_k in RMWP with two mandatory*

parts on uniprocessors is

$$OD_k^1 = \max(0, D_k - m_k^2 - \sum_{i<k} \left\lceil \frac{T_k}{T_i} \right\rceil m_i),$$

Global Rate Monotonic with Wind-up Part (G-RMWP) [3] is a global semi-fixed-priority scheduling algorithm on multiprocessors. Global scheduling permits to migrate tasks from one processor to another processor at run-time. Like RMWP, G-RMWP is based on and extends Global Rate Monotonic (G-RM) [1]. The optional deadline of each task in G-RMWP with two mandatory parts on multiprocessors is calculated by the following theorem.

Theorem 2 (From literature [3])**.** *The first relative optional deadline OD_k^1 of task τ_k in G-RMWP with two mandatory parts on multiprocessors is*

$$OD_k^1 = \begin{cases} \max(0, D_k - m_k^2) & (k \leqq M) \\ \max(0, D_k - m_k^2 - \hat{I}_k) & (k > M), \end{cases}$$

where $\hat{I}_k$ is the sum of the worst case interference time of task τ_k interfered by higher priority tasks $\tau_i (i < k)$. Due to the complexity of $\hat{I}_k$, the detail of $\hat{I}_k$ is shown in literature [3].

Unfortunately, these algorithms do not support multiple mandatory parts so that they are not adapted to a primary target application with multiple mandatory parts, as shown in Subsection 2.1.

4. Semi-Fixed-Priority Scheduling with Multiple Mandatory Parts

This paper presents semi-fixed-priority scheduling with multiple mandatory parts on uniprocessors and multiprocessors. Thanks to multiple mandatory parts, a primary target application as shown in Subsection 2.1 can be supported. In addition, multiprocessor real-time scheduling for imprecise computation can also be supported to overcome overloaded conditions. In particular, we present how to calculate optional deadlines of each task in RMWP and G-RMWP with multiple mandatory parts. The schedulability analysis shows that both RMWP and G-RMWP strictly dominates both RM and G-RM respectively. Finally we show an example of schedule generated by RMWP and G-RMWP with multiple mandatory parts.

4.1 RMWP Algorithm with Multiple Mandatory Parts

First of all, this paper introduces how to calculate optional deadlines of each task in RMWP with multiple mandatory parts on uniprocessors.

Theorem 3 (Optional deadline in RMWP algorithm)**.** *The l^{th} relative optional deadline OD_k^l of task τ_k with multiple mandatory parts in RMWP on uniprocessors is*

$$OD_k^l = \begin{cases} \max(0, D_k - m_k^{n_k^m} - \sum_{i<k} \left\lceil \frac{T_k}{T_i} \right\rceil m_i & (l = n_k^o) \\ \max(0, OD_k^{l+1} - m_k^{l+1} - o_k^{l+1}) & (l < n_k^o). \end{cases}$$

Proof. If $l = n_k^o$, it is clear that if task τ_k completes its mandatory part by its optional deadline, task τ_k completes the following mandatory part by its following optional deadline or deadline by theorem 1. If $l < n_k^o$, then the l^{th} relative optional deadline OD_k^l meets the definition of the optional deadline. Because the $l + 1^{th}$ optional deadline OD_k^{l+1} is subtracted from both the $l + 1^{th}$ mandatory part and the $l+1^{th}$ optional part. Hence, this theorem holds. □

By theorem 3, RMWP supports the practical imprecise task with multiple mandatory parts. Next this paper analyzes the schedulability of RMWP.

Theorem 4 (RMWP algorithm strictly dominates RM algorithm)**.** *RMWP strictly dominates RM. That is to say, one task set is schedulable by RMWP with multiple mandatory parts if the task set is schedulable by RM.*

Proof. If all optional deadlines are equal to 0, RMWP generates the same schedule as RM. In this case, the worst case interference time occurs in RMWP. Hence, RMWP strictly dominates RM in a similar way [2]. □

Theorem 5 (Least upper bound of RMWP algorithm)**.** *For a set of n tasks with semi-fixed-priority assignment, the least upper bound of RMWP with multiple mandatory parts on uniprocessors is $U_{lub} = n(2^{1/n} - 1)$.*

Proof. By theorem 4, RMWP strictly dominates RM. Hence, the least upper bound of RMWP on uniprocessors is equal to that of RM [10]. □

By theorem 5, the least upper bound of RMWP for n tasks is proved. In contrast, the least upper bound of RMWP for two tasks can be excluded thanks to theorem 5. Therefore, this theorem is shown in Appendix.

Figure 4 shows an example of schedule generated by RMWP and RM on a uniprocessor. Each task has three mandatory parts ($n_i^m = 3$) and two optional parts ($n_i^o = 2$). The task set is shown in Table 1. The optional deadlines of each task are calculated by theorem 3. In RM, each task does not execute optional parts because the overrun of the optional part may miss its deadline so that each task executes all mandatory parts continuously. In RMWP, each task can execute its optional parts if there is no task, which is ready to execute its mandatory parts. In addition, RMWP strictly dominates RM by theorem 4. Therefore, RMWP has the above advantages against RM.

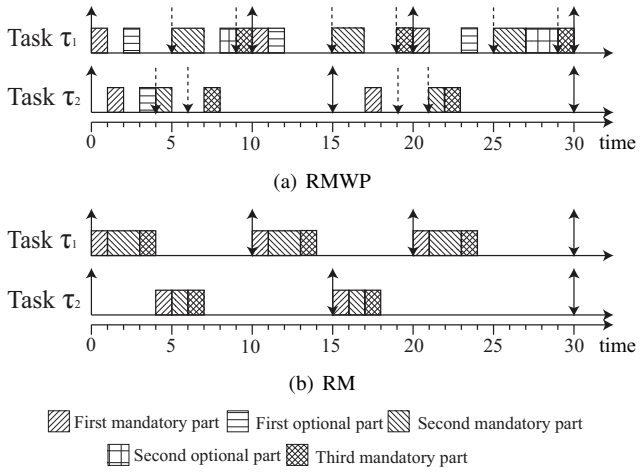

Figure 4. An example of schedule on a uniprocessor

Table 1. Task set A

Task	T_i	D_i	OD_i^1	OD_i^2	m_i^1	m_i^2	m_i^3	o_i^1	o_i^2
τ_1	10	10	5	9	1	2	1	1	2
τ_2	15	15	4	6	1	1	1	1	1

4.2 G-RMWP Algorithm with Multiple Mandatory Parts

First of all, this paper explains how to calculate optional deadlines of each task in G-RMWP with multiple mandatory parts.

Theorem 6 (Optional deadline in G-RMWP algorithm)**.** *The l^{th} relative optional deadline OD_k^l of task τ_k with multiple mandatory parts in G-RMWP on multiprocessors is the following equation.*

$$OD_k^l = \begin{cases} \max(0, D_k - m_k^{n_k^m}) & (k \leqq M \text{ and } l = n_k^o) \\ \max(0, D_k - m_k^{n_k^m} - \hat{I}_k) & (k > M \text{ and } l = n_k^o) \\ \max(0, OD_k^{l+1} - m_k^{l+1} - o_k^{l+1}) & (l < n_k^o) \end{cases}$$

Here, $\hat{I}_k$ is too complex to explain so that the detail of $\hat{I}_k$ is shown in literature [3].

Proof. If $k \leqq M$ and $l = n_k^o$, it is clear that the following mandatory part of task τ_k does not miss its deadline if task τ_k completes its mandatory part at the l^{th} relative optional deadline OD_k^l. If $k > M$ and $l = n_k^o$, the worst case interference time of task τ_k by higher priority tasks $\tau_i (i < k)$ is at most $\hat{I}_k$ in theorem 2. In a similar way, if $l < n_k^o$, it is clear that task τ_k completes its following mandatory parts by its following optional deadlines or deadline if task τ_k completes its mandatory parts by its optional deadlines. Hence, this theorem holds. □

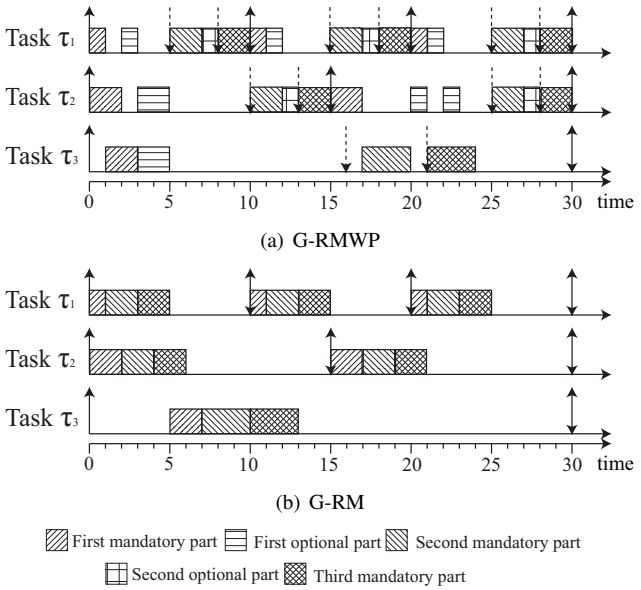

Figure 5. An example of schedule on two processors

Table 2. Task set B

Task	T_i	D_i	OD_i^1	OD_i^2	m_i^1	m_i^2	m_i^3	o_i^1	o_i^2
τ_1	10	10	5	8	1	2	2	1	1
τ_2	15	15	10	13	2	2	2	2	1
τ_3	30	30	16	21	2	3	3	2	2

By theorem 6, G-RMWP supports the practical imprecise task with multiple mandatory parts. Next this paper analyzes the schedulability of G-RMWP.

Theorem 7 (G-RMWP algorithm strictly dominates G-RM algorithm)**.** *G-RMWP strictly dominates G-RM. That is to say, one task set is schedulable by G-RMWP with multiple mandatory parts if the task set is schedulable by G-RM.*

Proof. If all optional deadlines are equal to 0, G-RMWP generates the same schedule as G-RM. In this case, the worst case interference time occurs in G-RMWP. Hence, G-RMWP strictly dominates G-RM in a similar way [3]. □

Theorem 8 (Least upper bound of G-RMWP algorithm)**.** *The least upper bound of G-RMWP with multiple mandatory parts on multiprocessors is*

$$U_{lub} = \frac{M}{2}(1 - U_{max}) + U_{max}, \tag{1}$$

where $U_{max} = \max\{U_i \mid i = 1, 2, 3, ..., n\}$.

Proof. By theorem 7, G-RMWP strictly dominates G-RM. Hence, the least upper bound of G-RMWP on multiprocessors is equal to that of G-RM [1]. □

Figure 5 shows an example of schedule generated by G-RMWP and G-RM on two processors. Each task has three mandatory parts ($n_i^m = 3$) and two optional parts ($n_i^o = 2$). The task set is shown in Table 2. The optional deadlines of each task are calculated by theorem 6. In G-RM, each task does not execute optional parts because the overrun of the optional part may miss its deadline so that each task executes all mandatory parts continuously. In G-RMWP, each task can execute its optional parts if there is no task, which is ready to execute its mandatory parts, as well as Figure 4. In addition, G-RMWP strictly dominates G-RM by theorem 7. Therefore, G-RMWP has the above advantages against G-RM.

5. Related Work

In this section, we show the researches of real-time scheduling for imprecise computation.

First of all, real-time scheduling algorithms in the imprecise computation model on multiprocessors are introduced. Khemka et al. discuss the problem of scheduling multiprocessors for imprecise computation, as a network flow problem [6]. Yun et al. propose a heuristic scheduling algorithm of imprecise computation with 0/1 constraint on multiprocessors [14]. Stavrinides and Karatza evaluate the performance of dynamic-priority scheduling in distributed real-time systems [12]. However, these approaches for multiprocessor or distributed systems do not analyze the schedulability. In contrast, our work has the theoretical contribution of multiprocessor imprecise computation including both optional deadline and schedulability analysis.

Next real-time scheduling algorithms in the practical imprecise computation model are introduced. M-FWP and SS-OP are dynamic-priority scheduling and support multiple mandatory parts on uniprocessors [7]. However, these algorithms do not support multiprocessors because the assignable time of optional parts are difficult to calculate on multiprocessors. In contrast, semi-fixed-priority scheduling supports multiprocessors in the practical imprecise computation model, thanks to the optional deadline [3]. Semi-fixed-priority scheduling is an effective technique for practical imprecise real-time applications on multiprocessors.

6. Concluding Remarks

This paper presents RMWP and G-RMWP with multiple mandatory parts on uniprocessors and multiprocessors respectively. In addition, this paper explains how to calculate the optional deadlines of each task in RMWP and G-RMWP with multiple mandatory parts. The schedulability analysis shows that semi-fixed-priority scheduling strictly dominates fixed-priority scheduling regardless of the number of mandatory parts. Moreover, an example of schedule in RMWP and G-RMWP with multiple mandatory parts is described. Thanks to semi-fixed-priority scheduling with multiple mandatory parts, many imprecise real-time applications like a lip reading task by sensor fusion [13] can be supported.

In future work, we will implement a primary target application by sensor fusion in RT-Est [4], which is a real-time operating system for semi-fixed-priority scheduling algorithms.

References

[1] T. P. Baker. An Analysis of Fixed-Priority Schedulability on a Multiprocessor. *Real-Time Systems*, 32(1-2):49–71, 2006.

[2] H. Chishiro, A. Takeda, K. Funaoka, and N. Yamasaki. Semi-Fixed-Priority Scheduling: New Priority Assignment Policy for Practical Imprecise Computation. In *Proceedings of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 339–348, Aug. 2010.

[3] H. Chishiro and N. Yamasaki. Global Semi-fixed-priority Scheduling on Multiprocessors. In *Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 218–223, Aug. 2011.

[4] H. Chishiro and N. Yamasaki. RT-Est: Real-Time Operating System for Semi-Fixed-Priority Scheduling Algorithms. In *Proceedings of the 2011 International Symposium on Embedded and Pervasive Systems*, pages 358–365, Oct. 2011.

[5] iRobot Corporation. Roomba. `http://store.irobot.com/home/index.jsp`, accessed 2012-03-03.

[6] A. Khemka, R. Shyamasundar, and K. Subrahmanyam. Multiprocessors scheduling for imprecise computations in a hard real-time environment. In *Proceedings of the Seventh International Parallel Processing Symposium*, pages 374–378, 1993.

[7] H. Kobayashi. *REAL-TIME SCHEDULING OF PRACTICAL IMPRECISE TASKS UNDER TRANSIENT AND PERSISTENT OVERLOAD*. PhD thesis, Keio University, Mar. 2006.

[8] P. Kritsada and K. O. Yang. Sensor Fusion by Neural Network and Wavelet Analysis for Drill-Wear Monitoring. *Journal of Solid Mechanics and Materials Engineering*, 4(6):749–760, 2010.

[9] K. Lin, S. Natarajan, and J. W. S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In *Proceedings of the 8th IEEE Real-Time Systems Symposium*, pages 210–217, Dec. 1987.

[10] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20:46–61, 1973.

[11] A. Shintani, A. Ogihara, Y. Yamaguchi, Y. Hayashi, and K. Fukunaga. Speech Recognition Using HMM Based on Fusion of Visual and Auditory Information. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 77(11):1875–1878, 1994.

[12] G. L. Stavrinides and H. D. Karatza. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. *Journal of Systems and Software*, 83(6):1004–1014, 2010.

[13] K. Takahashi. Sensing System Integrating Audio and Visual Information : Concrete Examples of Sensor Fusion Systems. *Journal of the Institute of Electronics, Information, and Communication Engineers*, 79(2):155–161, 1996.

[14] K. Yun, K. Song, K. Choi, G. Jung, S. Park, M. Hong, and D. Choi. A Heuristic Scheduling Algorithm of Imprecise Multiprocessor System with 0/1 Constraint. In *Proceedings of the Third International Workshop on Real-Time Computing Systems Application*, pages 307–313, Oct. 1996.

Appendix - Schedulability Analysis of RMWP Algorithm for Two Tasks

This appendix analyzes the least upper bound of RMWP with multiple mandatory parts for two tasks on uniprocessors in Subsection 4.1.

Theorem 9 (Least upper bound of RMWP algorithm for two tasks)**.** *For a set of two tasks with semi-fixed-priority assignment, the least upper bound of RMWP with multiple mandatory parts on uniprocessors is* $U_{lub} = 2(2^{1/2} - 1)$.

Proof. As before $F = \lfloor T_2/T_1 \rfloor$ be the number of periods of task τ_1 entirely contained in T_2. Without loss of generality, the computation time $\sum_{L=1}^{n_2^m} m_2^L$ is adjusted to fully utilize the processors. Now we consider four cases of schedulability analysis of RMWP as shown in Figure 6.

Case 1: As shown in Figure 6(a), when job $\tau_{2,2}$ is released, task τ_1 executes the l^{th} mandatory part and there is no idle processor time between mandatory parts of task τ_1. In this case, the equation $T_2 - T_1F \leqq \sum_{L=1}^{n_1^m} m_1^L$ is met so that the maximum of all mandatory parts of task τ_2 is

$$\sum_{L=1}^{n_2^m} m_2^L = (T_1 - \sum_{L=1}^{n_1^m} m_1^L)F.$$

The corresponding upper bound U_{ub} is

$$\begin{aligned} U_{ub} &= \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} + \frac{\sum_{L=1}^{n_2^m} m_2^L}{T_2} \\ &= \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} + \frac{(T_1 - \sum_{L=1}^{n_1^m} m_1^L)F}{T_2} \\ &= \frac{T_1}{T_2}F + \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} - \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_2}F \\ &= \frac{T_1}{T_2}F + \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_2}\left[\frac{T_2}{T_1} - F\right]. \end{aligned}$$

Since the quantity in square brackets is positive, the corresponding upper bound U_{ub} is monotonically increasing in $\sum_{L=1}^{n_1^m} m_1^L$. Being $T_2 - T_1F \leqq \sum_{L=1}^{n_1^m} m_1^L$, the minimum of the corresponding upper bound U_{ub} occurs for

$$\sum_{L=1}^{n_1^m} m_1^L = T_2 - T_1F.$$

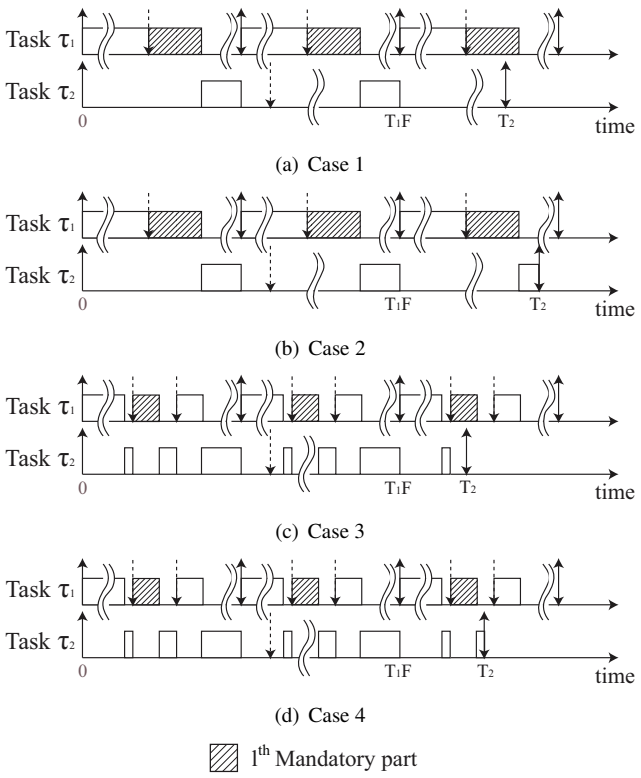

Figure 6. Four cases of schedulability analysis of RMWP

Case 2: As shown in Figure 6(b), when job $\tau_{2,2}$ is released, task τ_1 does not complete the l^{th} mandatory part by the l^{th} optional deadline and there is no idle processor time between mandatory parts of task τ_1. In this case, the equation $\sum_{L=1}^{n_1^m} m_1^L \leqq T_2 - T_1F$ is met so that the maximum of all mandatory parts of task τ_2 is

$$\sum_{L=1}^{n_2^m} m_2^L = T_2 - (F+1)\sum_{L=1}^{n_1^m} m_1^L.$$

The corresponding upper bound U_{ub} is

$$\begin{aligned} U_{ub} &= \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} + \frac{\sum_{L=1}^{n_2^m} m_2^L}{T_2} \\ &= \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} + \frac{T_2 - (F+1)\sum_{L=1}^{n_1^m} m_1^L}{T_2} \\ &= 1 + \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} - \frac{(F+1)\sum_{L=1}^{n_1^m} m_1^L}{T_2} \\ &= 1 + \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]. \end{aligned}$$

Since the quantity in square brackets is negative, the corresponding upper bound U_{ub} is monotonically increasing in $\sum_{L=1}^{n_1^m} m_1^L$. Being $\sum_{L=1}^{n_1^m} m_1^L \leqq T_2 - T_1F$, the minimum of the corresponding upper bound U_{ub} occurs for

$$\sum_{L=1}^{n_1^m} m_1^L = T_2 - T_1F.$$

Case 3: As shown in Figure 6(c), when job $\tau_{2,2}$ is released, task τ_1 executes the l^{th} mandatory part and there is processor time between mandatory parts of task τ_1. In this case, the equations $OD_1^l \leqq T_2 - T_1F \leqq OD_1^l + m_1^l$ and $0 \leqq \sum_{L=l+1}^{n_i^m} m_1^L \leqq T_1 - OD_1^{l+1}$ are met so that the maximum of all mandatory parts of task τ_2 is

$$\sum_{L=1}^{n_2^m} m_2^L = (T_1 - \sum_{L=1}^{n_1^m} m_1^L)F + OD_1^l - \sum_{L=1}^{l-1} m_1^L.$$

The corresponding upper bound U_{ub} is

$$\begin{aligned} U_{ub} &= \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} + \frac{\sum_{L=1}^{n_2^m} m_2^L}{T_2} \\ &= \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} + \frac{(T_1 - \sum_{L=1}^{n_1^m} m_1^L)F + OD_1^l - \sum_{L=1}^{l-1} m_1^L}{T_2} \\ &= \frac{T_1}{T_2}F + \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_2}\left[\frac{T_2}{T_1} - F\right] + \frac{OD_1^l - \sum_{L=1}^{l-1} m_1^L}{T_2} \\ &= \frac{T_1}{T_2}F + \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_2}\left[\frac{T_2}{T_1} - F\right] \\ &\quad + \frac{OD_1^l + m_1^l - \sum_{L=1}^{l} m_1^L}{T_2}. \end{aligned}$$

Since the quantity in square brackets is positive, the corresponding upper bound U_{ub} is monotonically increasing in $\sum_{L=1}^{n_1^m} m_1^L$. In addition, $OD_1^l + m_1^l - \sum_{L=1}^{l} m_1^L$ is positive so that the minimum of that equation occurs if $OD_1^l + m_1^l - \sum_{L=1}^{l} m_1^L = 0$. That is to say, $OD_1^l + m_1^l = \sum_{L=1}^{l} m_1^L$. Being $OD_1^l \leqq T_2 - T_1F \leqq OD_1^l + m_1^l$ and $0 \leqq \sum_{L=l+1}^{n_i^m} m_1^L \leqq T_1 - OD_1^{l+1}$, the minimum of the corresponding upper bound U_{ub} occurs for

$$\sum_{L=1}^{n_1^m} m_1^L = \sum_{L=1}^{l} m_1^L + \sum_{L=l+1}^{n_i^m} m_1^L = T_2 - T_1F.$$

Case 4: As shown in Figure 6(d), when job $\tau_{2,2}$ is released, task τ_1 completes the l^{th} mandatory part by the l^{th} optional deadline and does not start to execute the $l+1^{th}$ mandatory part. In addition, there is processor time between mandatory parts of task τ_1. In this case, the equations $0 \leqq \sum_{L=1}^{l} m_1^L \leqq T_2 - T_1F \leqq OD_1^l$ and $0 \leqq \sum_{L=l+1}^{n_i^m} m_1^L \leqq T_1 - OD_1^{l+1}$ are met so that the maximum of all mandatory parts of task τ_2 is

$$\sum_{L=1}^{n_2^m} m_2^L = T_2 - [(F+1)\sum_{L=1}^{l} m_1^L + F\sum_{L=l+1}^{n_i^m} m_1^L].$$

The corresponding upper bound U_{ub} is

$$\begin{aligned} U_{ub} &= \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} + \frac{\sum_{L=1}^{n_2^m} m_2^L}{T_2} \\ &= \frac{\sum_{L=1}^{n_1^m} m_1^L}{T_1} + \frac{T_2 - [(F+1)\sum_{L=1}^{l} m_1^L + F\sum_{L=l+1}^{n_i^m} m_1^L]}{T_2} \\ &= \left(\frac{1}{T_1} - \frac{F}{T_2}\right)\sum_{L=l+1}^{n_i^m} m_1^L + 1 + \frac{\sum_{L=1}^{l} m_1^L}{T_1} \\ &\quad - \frac{(F+1)\sum_{L=1}^{l} m_1^L}{T_2}. \end{aligned}$$

After that, the corresponding upper bound U_{ub} is partially differentiated by $\sum_{L=l+1}^{n_i^m} m_1^L$.

$$\frac{\partial U_{ub}}{\partial \sum_{L=l+1}^{n_i^m} m_1^L} = \frac{1}{T_1} - \frac{F}{T_2} = \frac{1}{T_2}\left[\frac{T_2}{T_1} - F\right]$$

Since the quantity in square brackets is positive, the corresponding upper bound U_{ub} is monotonically increasing. In this case, the minimum of the corresponding upper bound U_{ub} is as the following equation if $\sum_{L=l+1}^{n_i^m} m_1^L = 0$.

$$\begin{aligned} U_{ub} &= 1 + \frac{\sum_{L=1}^{l} m_1^L}{T_1} - \frac{(F+1)\sum_{L=1}^{l} m_1^L}{T_2} \\ &= 1 + \frac{\sum_{L=1}^{l} m_1^L}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right] \end{aligned}$$

Since the quantity in square brackets is negative, the corresponding upper bound U_{ub} is monotonically decreasing in $\sum_{L=1}^{l} m_1^L$. Being $0 \leqq \sum_{L=1}^{l} m_1^L \leqq T_2 - T_1F \leqq OD_1^l$ and $0 \leqq \sum_{L=l+1}^{n_i^m} m_1^L \leqq T_1 - OD_1^{l+1}$, the minimum of the corresponding upper bound U_{ub} occurs for

$$\sum_{L=1}^{n_1^m} m_1^L = \sum_{L=1}^{l} m_1^L + \sum_{L=l+1}^{n_i^m} m_1^L = T_2 - T_1F.$$

The upper bound of the corresponding upper bound U_{ub} occurs if $\sum_{L=1}^{n_1^m} m_1^L = T_2 - T_1F$ in all cases. This value is equal to that of Theorem 3 in [10]. Hence, the least upper bound of RMWP for two tasks on uniprocessors is

$$U_{lub} = 2(2^{1/2} - 1). \tag{2}$$

□