Scheduling Aperiodic Tasks using Total Bandwidth Server on Multiprocessors *

Shinpei Kato and Nobuyuki Yamasaki
Department of Information and Computer Science
Keio University, Yokohama, Japan
{shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract

This paper presents real-time scheduling techniques for reducing the response time of aperiodic tasks scheduled with real-time periodic tasks on multiprocessor systems. Two problems are addressed in this paper: (i) the scheduling of aperiodic tasks that can be dispatched to any processors when they arrive, and (ii) the scheduling of aperiodic tasks that must be executed on particular processors on which they arrive. In order to improve the responsiveness to both types of aperiodic tasks, efficient dispatching and migration algorithms are designed based on the Earliest Deadline First (EDF) algorithm and the Total Bandwidth Server (TBS) algorithm. The effectiveness of the designed algorithms is evaluated through simulation studies.

Keywords

Real-Time Scheduling, Response Time, Earliest Deadline First, Total Bandwidth Server, Multiprocessors.

1 Introduction

Over the years, real-time systems have focused on a periodic computation model, in which every task is invoked recurrently at certain interval. In fact, many types of embedded applications follow the periodic computation model. The scheduling of periodic tasks has therefore been one of the main subjects in the real-time computing community. Today, it is widely known that the Earliest Deadline First (EDF) algorithm [8] is optimal for the scheduling of recurrent real-time tasks on single core platforms.

In modern real-time systems, the computation model is more complex. For instance, embedded systems must operate in dynamic environments where human activities occur at any moment, then some tasks, such as emergency task, external event task, human interaction task, etc., arrive *aperiodically*. Contrary to periodic tasks, most of those aperiodic tasks are required to complete as early as possible rather than to complete by deadlines.

The most simplified technique for handling aperiodic tasks is to allocate available time slots that are left unused by periodic tasks. Such a background scheduling is fairly straightforward, however the sufficient responsiveness is never acquired. In order to reduce the response time of aperiodic tasks, a system typically carries out an *aperiodic server*. The aperiodic server is a special periodic task whose purpose is to service aperiodic requests as soon as possible. Like any periodic task, a server is characterized by a period and a fixed execution time called server capacity. The server is scheduled with the same algorithm used for the periodic tasks, and once active, it serves the aperiodic tasks within the range of its capacity.

For classical single-processor systems, a lot of efficient server algorithms have been developed. Lehoczky *et al.* originally developed fixed-priority server algorithms called Deferrable Server (DS) and Priority Exchange [7]. Sprunt *et al.* then extended these algorithms in the manner of capacity preservation and replenishment to Sporadic Server (SS) and Extended Priority Exchange (EPE) [10, 11]. Spuri *et al.* developed several dynamic-priority server algorithms based on the above techniques [12, 13]. According to their simulation results, the Total Bandwidth Server (TBS) algorithm showed excellent performance in response time, while its design is fairly simplified. For multiprocessor systems, on the other hand, it is surprising that few server techniques have been developed, though the use of multicore processors is now commonplace.

This paper presents real-time scheduling techniques for reducing the response time of aperiodic tasks scheduled with periodic tasks on multiprocessor systems. Since the system has multiple processors, system designers may consider various aperiodic task models. Thus, two problems are addressed in this paper: (i) the scheduling of aperiodic tasks that can be dispatched to any processors when they ar-

*This work is supported by the fund of Research Fellowships and Global COE Program of the Japan Society for the Promotion of Science for Young Scientists. This work is also supported in part by the fund of Core Research for Evolutional Science and Technology, Japan Science and Technology Agency.

978-0-7695-3492-3/08 $25.00 © 2008 IEEE
DOI 10.1109/EUC.2008.28

rive, and (ii) the scheduling of aperiodic tasks that must be executed on particular processors on which they arrive. We believe that the solutions to these two problems are also applicable to other combined cases. Taking into account the balance between performance and simplicity, the scheduling techniques are designed based on EDF and TBS.

The rest of this paper is organized as follows. The prior work is described in the next section. The system model and the terminology are defined in Section 3. Section 4 then presents the scheduling techniques. In Section 5, the effectiveness of the presented techniques is evaluated by several sets of simulations. This paper is concluded in Section 6.

2 Prior Work

Andersson *et al.* considered the real-time scheduling of aperiodic tasks with deadlines on multiprocessors [1, 2]. They improved the guaranteed real-time performance in both global scheduling and partitioned scheduling. The global scheduling algorithm developed in their work can always meet all deadlines, if the system utilization is less than 50%. The partitioned scheduling algorithm, on the other hand, can always meet all deadlines if the system utilization is less than 31%.

Baruah *et al.* considered a multiprocessor implementation of TBS [5]. They designed a scheduling algorithm based on EDF and TBS, which provides guaranteed real-time performance to aperiodic tasks on multiprocessors by the schedulability test.

Since the above work enabled the guarantee of the real-time performance for aperiodic tasks on multiprocessors, their contribution was remarkable. However, their goal was to satisfy the timing constraints of aperiodic tasks rather than to improve the responsiveness.

Banus *et al.* developed the Dual Priority algorithm, which was originally invented by Davis [6] to improve the responsiveness of fixed-priority scheduling on single-processors, to achieve low mean response time of aperiodic tasks while all periodic tasks are guaranteed to meet their deadlines in the global fixed-priority scheduling scheme [3]. Though the Dual Priority algorithm requires recursive computations at runtime, it was shown that the response time can be reduced very much especially when the system is heavily loaded.

To the best of our knowledge, no prior work have focused on reducing the response time of aperiodic tasks in the partitioned EDF scheduling scheme. However, most of commercial real-time systems prefer the partitioned approach rather than the global approach due to the simplicity of design. In addition, EDF has advantage in available system utilization, compared to fixed-priority algorithms. Thus, we address the scheduling of real-time systems in which tasks are partitioned among processors and scheduled by EDF.

3 System Model

The system is a multicore system composed of M processors (cores): $P_1, P_2, ..., P_M$. The code and data of programs (tasks) are shared among the processors. The overhead of inter-processor communication and task migration is not handled in this paper. In fact, the cost of inter-processor communication is not expensive in the modern multicore architecture. Besides, the cost of task migration can be ignorable by the context cache technique [15]. However, more or less, the overhead is still generated by these actions. Thus, we will address this problem in the future work.

A periodic task set, denoted by $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, is given to the system. The ith periodic task is defined by $\tau_i(C_i, T_i)$ where C_i is a worst-case execution time and T_i is a period. The processor utilization of τ_i is denoted by $U_i = C_i/T_i$. A task generates a sequence of jobs periodically. The jth job of τ_i is denoted by $\tau_{i,j}$, which is released at time $r_{i,j}$ and has a deadline of $d_{i,j}$ equal to the release time of the next job, i.e. $d_{i,j} = r_{i,j} + T_i$. The remaining execution time of $\tau_{i,j}$ at time t is expressed by $\tilde{c}_{i,j}(t)$. The total utilization of Γ is denoted by $U(\Gamma) = \sum_{i=1}^{n} U_i$.

Aperiodic tasks arrive sequentially. The kth aperiodic task is defined by $\alpha_k(a_k, E_k)$ where a_k is its arrival time and E_k is its worst-case execution time. Letting f_k be the completion time of α_k, the response time of α_k is expressed by $R_k = f_k - a_k$. For all k, a condition of $a_k < a_{k+1}$ is satisfied. The aperiodic load $U(\alpha)$ is determined by the average service rate μ and the average arrival rate λ, i.e. $U(\alpha) = \lambda/\mu$. An invocation of each aperiodic task is assumed to follow the Poisson arrival model in this paper.

Each periodic task is assigned to a particular processor and scheduled by EDF. The total utilization of each processor P_x is denoted by $U(P_x)$. All the tasks are preemptive and independent. Each processor does not execute more than one task at the same time. The same job is not executed in parallel. Since the scheduling of periodic tasks relies on the worst-case execution times, the performance deterioration due to transient degradation of the cache hit ratio caused by task migrations is not taken into account.

4 Scheduling Techniques

This section presents two scheduling techniques for reducing the response time of aperiodic tasks on multiprocessors. The first technique is designed for the scheduling of aperiodic tasks that can be executed on any processor. Then, the second technique is designed for the scheduling of aperiodic tasks that must be executed on a particular processor. In this section, we begin with the explanation of TBS to make the presented techniques more comprehensible. We then describe the details of the presented techniques.

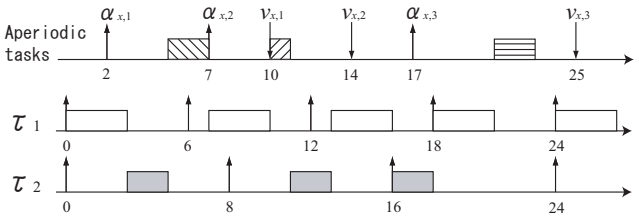

Figure 1. Example of Total Bandwidth Server

4.1 Total Bandwidth Server Review

Every time an aperiodic task arrives, the TBS algorithm assigns the possible earliest deadline to the aperiodic task. Since the aperiodic task assumed in this paper does not have a deadline, it is a virtual deadline. Once the task is assigned the virtual deadline, it is scheduled by EDF with periodic tasks. The virtual deadline is determined based on the server bandwidth (utilization). Suppose that processor P_x carries out server τ_x^{srv} with bandwidth U_x^{srv}. When the kth aperiodic task, denoted by $\alpha_{x,k}$, arrives at time $a_{x,k}$ with execution time $E_{x,k}$, its virtual deadline $v_{x,k}$ is calculated by Equation (1), where $v_{x,0} = 0$ by definition.

$$v_{x,k} = \max\{a_{x,k}, v_{x,k-1}\} + \frac{E_{x,k}}{U_x^{srv}} \quad (1)$$

Figure 1 shows an example of EDF scheduling on P_x produced by two periodic tasks, $\tau_1(3, 6)$ and $\tau_2(2, 8)$, and a server with bandwidth $U_x^{srv} = 1 - (U_1 + U_2) = 0.25$. The first aperiodic task $\alpha_{x,1}(2, 2)$ arrives at time $a_{x,1} = 2$ with execution time $E_{x,1} = 2$ and is serviced with virtual deadline $v_{x,1} = a_{x,1} + E_{x,1}/U_x^{srv} = 2 + 2/0.25 = 10$. Since $v_{x,1}$ is later than the deadlines of $\tau_{1,1}$ and $\tau_{2,1}$, $a_{x,1}$ begins execution after $\tau_{1,1}$ and $\tau_{2,1}$ complete. Similarly, the second aperiodic task $\alpha_{x,2}(7, 1)$, which arrives at time $a_{x,2} = 7$ with execution time $E_{x,2} = 1$, is assigned virtual deadline $v_{x,2} = \max\{a_{x,2}, v_{x,1}\} + E_{x,2}/U_x^{srv} = 10 + 4 = 14$ but is not serviced immediately, since τ_1 is still active with an earlier deadline of 12. Finally, the third aperiodic task $\alpha_{x,3}(17, 2)$ arrives at time $a_{x,3} = 17$ with execution time $E_{x,3} = 2$ and receives virtual deadline $v_{x,3} = a_{x,3} + E_{x,3}/U_x^{srv} = 25$. It is not executed immediately, since τ_1 and τ_2 have deadlines earlier than $\alpha_{x,3}$.

4.2 Dispatching Technique

This section considers the scheduling of aperiodic tasks that can be executed on any processor. It is easy to reduce the response time of aperiodic tasks in this case. We can reduce the response time by dispatching an arriving aperiodic task to a processor on which the response time of the task is expected to be minimized. The concept of the dispatching technique is depicted in Figure 2. The concern here is

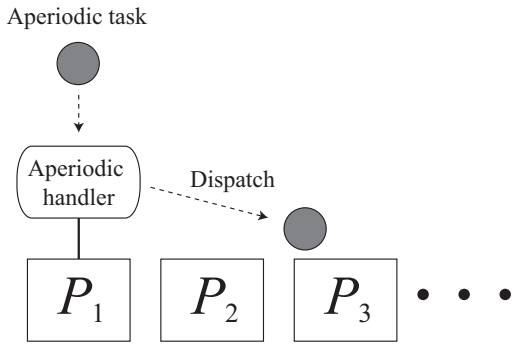

Figure 2. Concept of dispatching technique

how to estimate the response time. Unlike the fixed-priority server algorithms, it is complicated to analyze the worst-case response time of an aperiodic task in TBS, since the priority order of tasks is dynamically changed. Therefore, another indicator of the response time is required.

The design of a dispatching technique presented in this paper is fairly simple. As we already described in Section 4.1, TBS assigns a virtual deadline to every aperiodic task. Then, aperiodic tasks are scheduled to complete by their virtual deadlines. Thus, the virtual deadline can be deemed to be the worst-case response time, though the aperiodic task is expected to complete much earlier than the virtual deadline. However, it can complete just at the virtual deadline in the worst case. So, it is expected that the response time of the aperiodic task is minimized by dispatching it to a processor on which the earliest virtual deadline is assigned. In consequence, the aperiodic task handler is designed so that it dispatches an arriving aperiodic task to a processor on which the task has the minimum value of Equation (1).

4.3 Migration Technique

This section considers the scheduling of aperiodic tasks that must be executed on a particular processor. In this case, the use of periodic task migrations is effective to reduce the response time of aperiodic tasks. The concept of migration technique is depicted in Figure 3. The execution of an aperiodic task pending on P_1 is waited by two periodic tasks with earlier deadlines. If the two periodic tasks are migrated to other processors without timing violations, the aperiodic task can receive immediate service and the response time can be reduced.

Henceforth, the kth aperiodic task arriving on P_x is denoted by $\alpha_{x,k}(a_{x,k}, E_{x,k})$, where $a_{x,k}$ is its arrival time and $E_{x,k}$ is its worst-case execution time, estimated in advance. The virtual deadline assigned to $\alpha_{x,k}$ is described by $v_{x,k}$. Then, the finish time of $\alpha_{x,k}$ is defined by $f_{x,k}$ and the response time of $\alpha_{x,k}$ is defined by $R_{x,k} = f_{x,k} - a_{x,k}$.

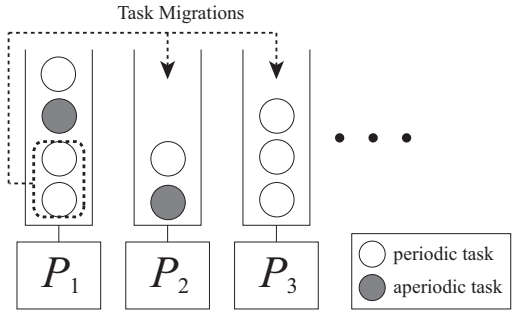

Figure 3. Concept of migration technique

Suppose that at time t any aperiodic task $\alpha_{x,k}$ arrives on any P_x, which has its own server with bandwidth U_x^{srv}, and is assigned a virtual deadline $v_{x,k}$ obtained by Equation (1). At this moment, the presented technique exploits temporal migrations according to the following procedure.

1. Seek a periodic job $\tau_{i,j}$ with the earliest deadline, which has not been already migrated in its period. If such a job does not exist, the procedure exits.

2. Seek a processor P_y that satisfies Inequation (2) where $v_{y,l}$ is the virtual deadline assigned to the last aperiodic task $\alpha_{y,l}$ that arrived on P_y, and $\tilde{c}_{i,j}(t)$ is the remaining execution time of $\tau_{i,j}$.

$$d_{i,j} \geq \max\{t, v_{y,l}\} + \frac{\tilde{c}_{i,j}(t)}{U_y^{srv}} \tag{2}$$

3. Migrate $\tau_{i,j}$ to P_y and virtually deals with it as the $l+1$th aperiodic task on P_y with virtual deadline $v_{y,l+1}$, expressed by Equation (3). Note that the next aperiodic task arriving on P_y after time t is indexed by $l+2$.

$$v_{y,l+1} = \max\{t, v_{y,l}\} + \frac{\tilde{c}_{i,j}(t)}{U_y^{srv}} \tag{3}$$

4. Reassign the improved virtual deadline $v'_{x,k}$, expressed by Equation (4), to $\alpha_{x,k}$.

$$v'_{x,k} = \max\{t, v_{x,k-1}\} + \frac{E_{x,k}}{U_x^{srv} + \frac{\tilde{c}_{i,j}(t)}{T_i}} \tag{4}$$

5. Migrate τ_i back to P_x at time $r_{i,j+1}$ for its next job $\tau_{i,j+1}$.

The procedure assumes that the migrating job $\tau_{i,j}$ transforms the aperiodic task $\alpha_{y,l+1}$. Hence, it must use $v_{y,l+1}$ to calculate the virtual deadline of the next arriving aperiodic task. Note that assigning $v_{y,l+1}$ to $\tau_{i,j}$ is not desirable because it is a periodic job and is not required to complete as soon as possible, thereby it had better to assign the true deadline $d_{i,j}$ so as to leave the available bandwidth as much as possible. However, such a deadline assignment incurs a complicated schedulability analysis. Thus, this paper takes a simpler policy in which the virtual deadline $v_{y,l+1}$ is assigned to the migrating job $\tau_{i,j}$. We will consider the schedulability analysis for the other policy in the future work. Notice that Step 1 of the procedure guarantees that any migrated job is never migrated again to another processor in the same period for practical use.

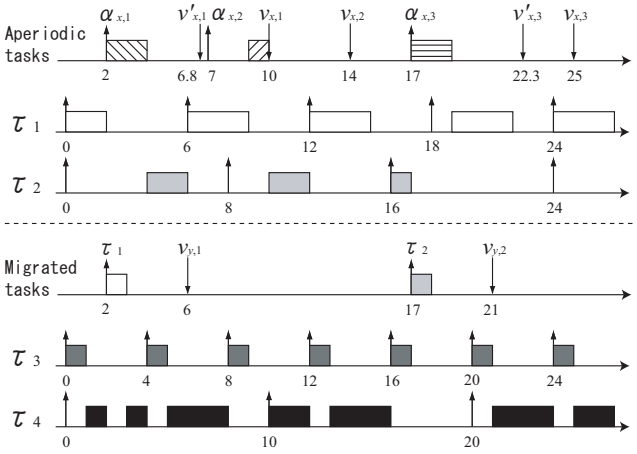

Figure 4. Example of temporal migrations

Figure 4 depicts an example of TBS with temporal migrations. Suppose that there are two processors, P_x and P_y. The same periodic tasks $\tau_1(3,6)$ and $\tau_2(2,8)$ in the example of Figure 1 are scheduled on P_x. In addition, other two tasks $\tau_3(1,4)$ and $\tau_4(5,10)$ are scheduled on P_y. The bandwidth of both servers on P_x and P_y is 0.25. For simplicity of explanation, we assume that no aperiodic tasks arrive on P_y. At time $t = 2$, the first aperiodic task $\alpha_{x,1}(2,2)$ arrives on P_x. Without migrations, it receives a virtual deadline of 10. Meanwhile, exploiting migrations, τ_1 can be temporarily migrated to P_y, since it satisfies Equation (2) and receives a virtual deadline $v_{y,1} = t + \tilde{c}_{1,1}(t)/U_y^{srv} = 2 + 1/0.25 = 6$ according to Equation (3). Then, $\alpha_{x,1}$ is assigned an improved virtual deadline $v'_{x,1} = 2 + 2/(0.25 + 1/6) = 6.8$ according to Equation (4). $\alpha_{x,1}$ gets an virtual deadline earlier than τ_2 and τ_1 is temporarily absent on P_x, $\alpha_{x,1}$ can be executed immediately. In contrast, τ_1 cannot be temporarily migrated when the second aperiodic task $\alpha_{x,2}$ arrives at time $t = 7$, since it would receive a virtual deadline $t + \tilde{c}_{1,2}(t)/U_y^{srv} = 7 + 2/0.25 = 15$ that is later than the original deadline 12. Hence, $\alpha_{x,2}$ does not receive an improved virtual deadline in this case. Finally, at time $t = 17$ the third aperiodic task $\alpha_{x,3}$ arrives and τ_2 can be migrated to P_y. $\alpha_{x,3}$ therefore receives an improved virtual deadline $v'_{x,3} = 17 + 2/(0.25 + 1/8) = 22.3$ and is serviced immedi-

ately, since it has a deadline earlier than τ_1 and τ_2. Comparing Figure 4 and Figure 1, the effectiveness of the presented migration technique can be clearly observed. The response time of $\alpha_{x,1}$ is reduced to $R_{x,1} = 4 - 2 = 2$ from $7 - 2 = 5$. Similarly, the response times of $\alpha_{x,2}$ and $\alpha_{x,3}$ are reduced to $R_{x,2} = 10 - 7 = 3$ and $R_{x,3} = 19 - 17 = 2$ from $11 - 7 = 4$ and $23 - 17 = 6$ respectively.

4.3.1 Responsiveness

Two benefits for reducing the response time of an aperiodic task are obtained by this migration policy. The first one is that the turn of the aperiodic execution comes faster, since a periodic job with an earlier deadline may be migrated to another processor, and hence it disappears from the processor. Such a benefit is observed in the response time of $\alpha_{x,2}$ in Figure 4. Although it does not an improved virtual deadline, but it can be executed one time unit earlier, because τ_1 disappears from the processor at time 2. The temporal migration procedure presented in Section 4.3 can be recursive to moreover improve the responsiveness to the arriving aperiodic task. More precisely, though the algorithm targets only one job with an earlier deadline in Step 1 of the procedure, if the algorithm chooses more than one job with an earlier deadline to be temporarily migrated to different processors, it is expected that the response time of the arriving aperiodic task can be moreover reduced. However, such a recursive procedure requires much more computation cost and is not adopted in the algorithm, because this work aims practical approaches.

The second benefit is that the server bandwidth is temporarily broaden due to temporal absences of migrating periodic jobs, which enables an aperiodic task to have an earlier virtual deadline as expressed by Equation (4). Such a benefit is observed in $\alpha_{x,1}$ and $\alpha_{x,3}$ in Figure 4. For example, $\tau_{x,1}$ is supposed to have a virtual deadline of 10 without temporal migrations, but it actually receives a virtual deadline of 6.8 with temporal migrations, and the improved virtual deadline skips over the deadline of τ_2 equal to 8. The same deadline skip can be seen for the case of $\alpha_{x,3}$. In addition, the responsiveness to the already-pending aperiodic tasks on P_y is never affected by a migrating job $\tau_{i,j}$, since the migration condition of Inequation (2) guarantees that the migrating job is inevitably assigned a virtual deadline later than any virtual deadlines of the pending aperiodic tasks. Therefore, the migrating job is never scheduled in advance of the pending aperiodic tasks, which means that the completion time of the pending aperiodic tasks is never prolonged. Meanwhile, the response time of the aperiodic tasks arriving in the future within $[t, v_{i,l+1}]$ on P_y can be worse unfortunately, because the virtual deadlines assigned to them are shifted for $v_{y,l+1} - v_{y,l}$ time units, compared to the ones without the temporal migration techniques. However, when there are periodic tasks with deadlines earlier than the assigned virtual deadlines of the arriving aperiodic tasks, the algorithm attempt temporal migrations after all, and the response time can be reduced as much as possible again. In consequence, the expectation is that the total responsiveness to aperiodic tasks is improved after all.

4.3.2 Schedulability

A migrating job $\tau_{i,j}$ and the original jobs scheduled on P_y are guaranteed to be schedulable. The proof is as follows. According to the TBS theory [12, 13], assigning virtual deadline $v_{y,l+1}$, which is earlier than its true deadline $d_{i,j}$, to $\tau_{i,j}$ on P_y does not jeopardize the schedulability of EDF. This fact implies that the original jobs are still guaranteed to be schedulable even though $\tau_{i,j}$ have true deadline $d_{i,j}$ according to the sustainability of EDF, which was analyzed by Baruah *et al.* [4]. As a result, the timing constraints of the periodic tasks are never broken by the presented migration technique.

4.3.3 Processor Selection

This section describes an efficient selection of the destination processor when more than one processor is obtained in Step 2 of the procedure. The processor must be determined so that the response time of aperiodic tasks can be reduced as much as possible. Here, the optimal selection of the destination processor is out of concern in this paper, because the selection is reduced to a bin-packing problem which is NP-hard in a strong sense to obtain the optimal result. Thus, the following heuristics are used in this paper to determine the destination processor.

- The First-Fit heuristic migrates $\tau_{i,j}$ to the first (i.e., lowest-indexed) processor that satisfies Inequation (2).
- The Best-Fit heuristic migrates $\tau_{i,j}$ to the processor that (i) satisfies Inequation (2), and (ii) minimizes the difference between $d_{i,j}$ and $v_{y,l+1}$.
- The Worst-Fit heuristic, on the other hand, migrates $\tau_{i,j}$ to the processor that (i) satisfies Inequation (2), and (ii) maximizes the difference between $d_{i,j}$ and $v_{y,l+1}$.

5 Simulation Studies

This section shows the effectiveness of the presented techniques by comparing with the pure TBS algorithm which does not exploit the presented techniques. The performance metric is the mean response time of aperiodic tasks in different loads. We first evaluate the dispatching technique. We then evaluate the migration technique.

5.1 Experimental Setup

A series of experimental evaluation makes use of the RmtSimulator [1], which is a MIPS-based CPU simulator for an SMT processor and a multicore processor. Since the simulator supports eight hardware threads (processing cores), the number of processors is chosen from 2, 4, and 8 in the experiments. Then, we have implemented the presented techniques in the TFlight kernel [2], which is a light-weight for the RmtSimulator. The scheduler exploits the EDF-FF algorithm [9] for the scheduling of periodic tasks.

A randomly-generated periodic task set with system utilization 60%, i.e. total utilization $0.6 \times M$, is submitted to each simulation. Task set Γ is generated as follows. A new periodic task is appended to Γ as long as $U(\Gamma)$ does not exceed $0.6 \times M$. The properties of each task refer to the applications implemented in the humanoid robot [14] which we have developed in the prior work. Each task utilizes the processor time within the range of $1 \sim 50\%$. Thus, the utilization of each periodic task is computed based on a uniform distribution within the range of $[0.01, 0.5]$, yet only the utilization of the task generated at the very end is adjusted so that $U(\Gamma)$ becomes equal to $0.6 \times M$. T_i is randomly computed within the range of $[100, 3000]$, since the feedback periods of control and multimedia tasks in the humanoid robot are set around $1 \sim 30ms$. Then, the execution time of each task is calculated by $C_i = U_i T_i$. All the utilization left unused by the periodic tasks is allocated to the bandwidth of a server on each processor P_x; that is, we have $U_x^{srv} = 1 - U(\Pi_x)$.

Aperiodic workloads are generated based on the M/M/1 queuing model. The arrival times of aperiodic tasks are calculated using a Poisson distribution, and the execution times are computed using an exponential distribution. The arrival processors for the aperiodic tasks are determined based on a uniform distribution for fairness. The average service rate is chosen from $\mu = 0.1$ and $\mu = 0.2$. The load of aperiodic tasks is varied across the range of system utilization unused by the periodic tasks. In particular, the range is from 0.1 to 0.4. Every aperiodic load, the mean response time is measured by the following formula.

$$\frac{\text{the sum of the response time of all aperiodic tasks}}{\text{the number of aperiodic tasks}}$$

5.2 Simulation Results

Figure 5 and Figure 6 show the improvement of the mean response time relative to the original TBS algorithm, achieved by the presented dispatching technique. According to the simulation results, the mean response time of aperiodic tasks is amazingly reduced. In particular, when the system load is not high, the improved amount of the mean response time is above 50 times. The mean response time is moreover improved as the number of processors is increased. In the best case, the improved amount of the mean response time reaches over 120 times when the number of processors is eight. Even though the system load is increased, the improved amount of the mean response time is around $2 \sim 12$ times. The mean response time is also influenced by the average service rate. According to the simulation results, the presented dispatching technique is more effective to the set of aperiodic tasks whose execution times are relatively short.

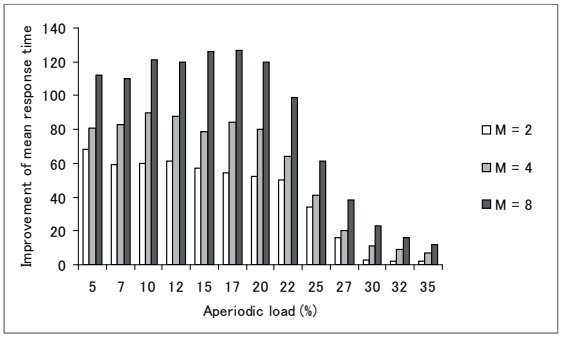

Figure 5. Performance of presented dispatching technique ($\mu = 0.1$).

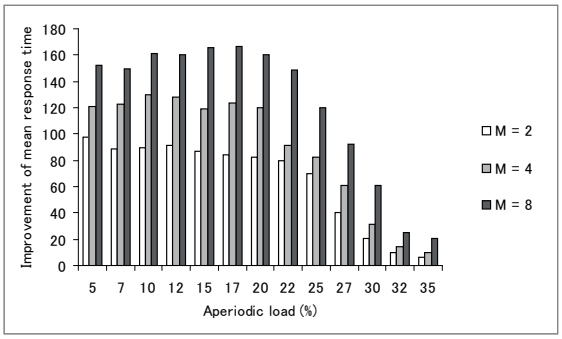

Figure 6. Performance of presented dispatching technique ($\mu = 0.2$).

We consider the factor of the performance improvement as follows. The priority of aperiodic tasks is determined based on the virtual deadline assigned by TBS. Hence, the aperiodic tasks can be never executed in advance to the periodic tasks with earlier deadlines. However, using the dispatching technique, each aperiodic task is dispatched to the processor so that it is assigned the earliest virtual deadline. Therefore, the response time of aperiodic tasks can be dramatically reduced by the dispatching technique. In conse-

[1] http://drtp.dip.jp/svn/RmtSimulator/
[2] http://drtp.dip.jp/svn/tflight

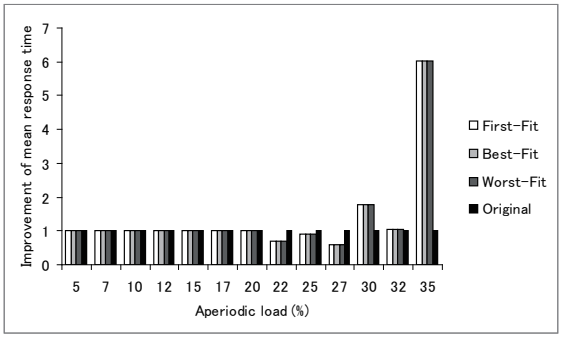

Figure 7. Performance of presented migration technique ($M = 2, \mu = 0.1$).

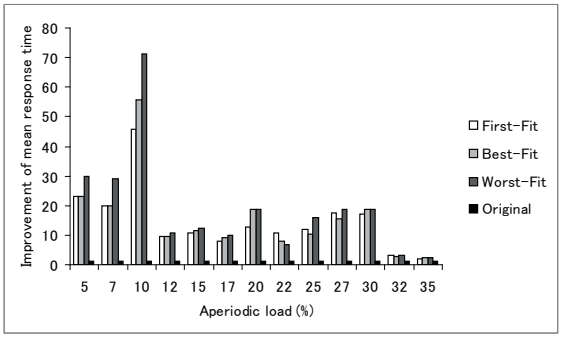

Figure 9. Performance of presented migration technique ($M = 8, \mu = 0.1$).

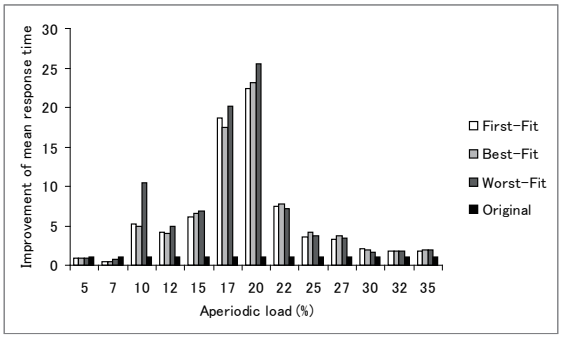

Figure 8. Performance of presented migration technique ($M = 4, \mu = 0.1$).

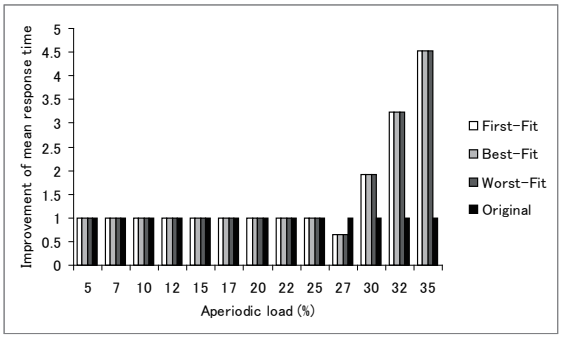

Figure 10. Performance of presented migration technique ($M = 2, \mu = 0.2$).

quence, the dispatching technique is fairly effective for the scheduling of aperiodic tasks that can be executed on any processors, since it can improve the responsiveness without complex computations such that each aperiodic task is only dispatched to a processor on which its response time is expected to be minimized.

Figure 7, Figure 8, and Figure 9 show the improvement of the mean response time relative to the original TBS algorithm, achieved by the presented migration technique, in the case of the average service rate 0.1. The improved amount of the mean response time is not as much as the dispatching technique, but we can still observe the effectiveness of the migration technique. When the system load is low, the original TBS can also make use of the processor time effectively. When the system load is high, on the other hand, the room for improvement of the response time does not exist very much. Thus, the presented migration technique is especially effective when the system load is moderate. However, when the number of processor is eight, the improved amount of the mean response time is remarkable even when the system load is low. Hence, we consider that the presented migration technique is more effective for systems supporting more processors. As for the selection of destination processors for migrations, it seems that the worst-fit heuristic relatively performs better than the other heuristics.

Figure 10, Figure 11, and Figure 12 show the improvement of the mean response time relative to the original TBS algorithm, achieved by the presented migration technique, with respect to the average service rate $\mu = 0.2$. The simulation results are similar to the previous case, though the maximum improved amount is slightly greater. Throughout the simulations, the number of task migrations per aperiodic task arrival was at most 0.18. Thus, we consider that the migration cost is not expensive.

6 Conclusion

This paper presented the real-time scheduling techniques for reducing the response time of aperiodic tasks on multiprocessor systems. We addressed (i) the scheduling of aperiodic tasks that can be dispatched to any processor when they arrive, and (ii) the scheduling of aperiodic tasks that must be executed on the processors on which they arrive. We first developed the dispatching technique that dispatches

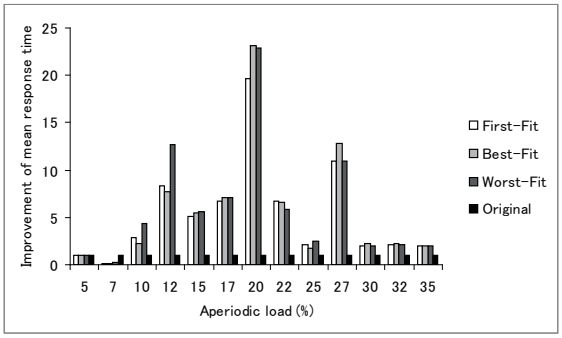

Figure 11. Performance of presented migration technique ($M = 4, \mu = 0.2$).

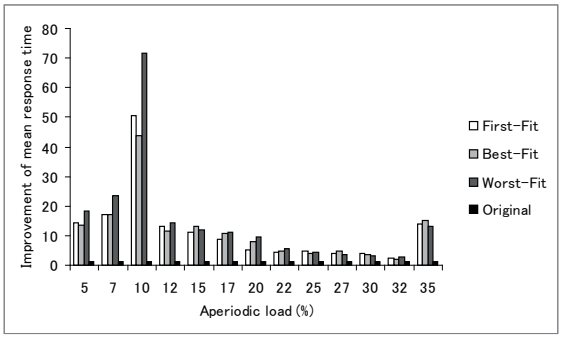

Figure 12. Performance of presented migration technique ($M = 8, \mu = 0.2$).

an arriving aperiodic task to the processor on which the response time of the task is expected to be minimized. We then developed the migration technique that temporarily migrates a periodic task with higher priority than an arriving aperiodic task to increase the available processor time for the aperiodic task. The responsiveness and schedulability are also discussed. In addition, we introduced the three heuristics for the selection of destination processors.

The simulation studies evaluated the effectiveness of the presented techniques. We showed that the mean response time of aperiodic task can be dramatically reduced by the presented techniques. Especially the effectiveness of the dispatching technique was remarkable, since it can improve performance dramatically while it can be realized easily such that each aperiodic task is only dispatching to the processor on which its response time is expected to be minimized. As for the migration technique, it was more effective when the Worst-Fit heuristics was applied for the selection of destination processors.

We will discuss the migration cost and consider precise theoretical analysis in the future work. We will also conduct experimental evaluation using more variety of parameters.

References

[1] B. Andersson, T. Abdelzaher, and J. Jonsson. Global Priority-Driven Aperiodic Scheduling on Multiprocessors. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2003.

[2] B. Andersson, T. Abdelzaher, and J. Jonsson. Partitioned Aperiodic Scheduling on Multiprocessors. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2003.

[3] J.M. Banus, A. Arenas, and J. Labarta. Dual Priority Algorithm to Scheduling Real-Time Tasks in a Shared Memory Multiprocessor. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2003.

[4] S.K. Baruah and A. Burns. Sustainable Scheduling Analysis. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 159–168, 2006.

[5] S.K. Baruah and G.Lipari. A Multiprocessor Implementation of the Total Bandwidth Server. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, pages 40–47, 2004.

[6] R.I. Davis and A. Wellings. Dual Priority Scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 100–109, 1995.

[7] J.P. Lehoczky, L. Sha, and J.K. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 261–270, 1987.

[8] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[9] J.M. Lopez, M. Carcia, J.L. Diaz, and D.F.Carcia. Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 25–33, 2000.

[10] B. Sprunt, J.P. Lehoczky, and L. Sha. Exploiting Unused Periodic Time for Aperiodic Service using the Extended Priority Exchange Algorithm. In *Proceeding of the IEEE Real-Time Systems Symposium*, pages 251–258, 1988.

[11] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Real-Time Systems*, 1(1):27–60, 1989.

[12] M. Spuri and G.C. Buttazo. Efficient Aperiodic Service under Earliest Deadline Scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 2–11, 1994.

[13] M. Spuri and G.C. Buttazo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real-Time Systems*, 10(2):179–210, 1996.

[14] T. Taira, N. Kamata, and N. Yamasaki. Design and Implementation of Reconfigurable Modular Robot Architecture. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3566–3571, 2005.

[15] N. Yamasaki. Responsive Multithreaded Processor for Distributed Real-Time Systems. *Journal of Robotics and Mechatronics*, 17(2):130–141, 2005.