Alternative Priority Scheduling in Dynamic Priority Systems*

Hyungill Kim †, Sungyoung Lee †, and Jongwon Lee † †

Department of Computer Engineering, Kyung Hee University, Seoul, Korea †
hikim@oslab.kyunghee.ac.kr, slee@nms.kyunghee.ac.kr
Software Research Lab., Korea Telecom, Seoul, Korea † †
jwlee@coral.kotel.co.kr

ABSTRACT

The major drawback of the slack-stealing based schedulings for aperiodic requests is a high computational complexity to calculate the slack which in consequence makes them not be practical. In this paper, we present a soft-aperiodic task scheduling algorithm, called Alternative Priority Scheduling (APS), which has a simple slack calculation method in dynamic priority systems. The proposed algorithm has been extended the EDF-CTI (Earliest Deadline First-Critical Task Indicating) Algorithm [6],[8] developed by the authors. The APS algorithm references the off-line built CTI table and chooses either an EDF or a CEF (Critical Execution time First) algorithm alternatively at run-time. This paper also demonstrates the optimality of the APS algorithm. Our simulation study shows that the APS algorithm, in most cases, is slightly better than the EDF-CTI algorithm and the other soft-aperiodic schedulings in terms of the short response time of aperiodic requests, and considerably improves the previous algorithms in a high workload.

1. Introduction

The problem of jointly scheduling hard deadline periodic tasks and soft deadline aperiodic tasks in real time systems is much more challenging than scheduling of periodic tasks alone. In the last few years considerable researches in the areas of jointly scheduling periodic and aperiodic tasks have been done in fixed priority systems [2],[14] such as the polling server, the bandwidth preservation [9],[17],[18] and the slack stealing algorithms [3],[10],[11]. On the other hands, less attention has been made to the same problems in the context of dynamic priority systems. Recently, Tia [19] introduced an optimal scheduling of aperiodic requests in dynamic priority systems based on the slack stealing approach. The main idea of their algorithm is to partition the periodic requests in sets such that the slack of all the requests in the same set is affected by the same scheduling events. Consequently, their algorithm can determine the minimum slack available at any time in *O(n)* time where *n* is the number of periodic tasks and is an optimal one in that it minimizes the response times of aperiodic requests. Homayoun and Ramanathan [5] extended the deferrable server scheduling algorithm to work with the EDF algorithm. The deferrable server algorithm, however, does not always fully utilize the processor due to the fact that the response times for aperiodic requests are sometimes not the minimum possible. Spuri and Buttazzo [15],[16] proposed four on-line schedulings for aperiodic requests in dynamic priority systems: the dynamic priority exchange, the total bandwidth, the EDL (Earliest Deadline as Late as possible) and the improved priority exchange algorithm. Although only the EDL algorithm among them is optimal, its run time overhead, however, is higher than that of Tia's algorithm.

In this paper, we present a new strategy for joint scheduling of soft-aperiodic and periodic tasks, called ***Alternative Priority Scheduling*** (APS) algorithm, which is optimal in dynamic priority systems. Some optimal schedulings for soft-aperiodic requests in dynamic priority systems already exist, so a natural question arises: why another? The answer is that none of the existing optimal schedulings satisfies the requirement of practicality in real-world applications due to their computational expenses for the slack calculation. Our goal, consequently, is to find another optimal scheduling for soft-aperiodic requests in the CTI (Critical Task Indicating) scheduling [7] framework, which preserves a

* This work was supported in part by the KOSEF under Grant No. 95-0100-07-01-3.

Authorized licensed use limited to: SAMSUNG ELECTRONICS DMC. Downloaded on March 16,2010 at 10:14:57 EDT from IEEE Xplore. Restrictions apply.

simple slack calculation scheme and well copes with the practicality.

The proposed algorithm has adopted a new type of scheduling scheme which chooses either an EDF or a CEF (Critical Execution time First) scheduling policy alternatively for a given periodic task set at run-time while referencing information in the CTI table [7] built off-line. The CTI table is created by the ***deadlinewise preassignment*** scheduling policy in which tasks are preassigned toward deadlines at their maximum. The scheduling information in the CTI table enables a scheduler to have a predictability and reduces the overall computational complexity at run time. Building a CTI table is very similar to the reverse schedule of the EDF introduced by Chetto and Chetto [1]. However, there are differences between the two approaches in terms of their applications and pursuing goals which are shown in [7]. Recall that the CTI algorithm [7] considers the problem of the joint scheduling of periodic and aperiodic tasks. The algorithm basically performs the on-line scheduling based on either a fixed-priority or a dynamic-priority scheduling while referencing the scheduling information in the off-line built CTI table.

The CEF scheduling gives the highest priority to a periodic task which has the largest amount of ***critical task's execution time (cet)*** at on-line. We define that a ***critical task*** is a periodic task which must be scheduled at certain time *t*, otherwise its deadline could not be guaranteed. We also define a ***cet*** as the time differences between all the computation requirement ($Cr_i(t)$) and all the computation processing completed ($Cp_i(t)$) in the CTI table for each periodic task until time *t*: ($Cr_i(t)-Cp_i(t)$). The ***cet*** is used to evaluate whether the preassigned periodic tasks in the CTI table must be executed or not to meet their deadlines. The major reason for introducing an alternative scheduling policy is to get a fast response time for aperiodic requests. We will discuss more details in subsection 4.2.

Meanwhile, the proposed algorithm newly introduces the concept of an ***optimistic*** slack calculation method rather than a ***minimal*** method which is used in the slack-stealing based schedulings. The term ***minimal*** method means that the slack stealing algorithm [10],[11] uses the actual slack as the minimum slack value of all periodic requests to ensure that all the lower priority periodic deadlines are met. On the other hands, the term ***optimistic*** means that a scheduler regards all the time as the slack except for the execution time of a given periodic task set to guarantee their deadlines. The ***optimistic*** approach therefore, is to maximize the reclaiming of unused computation time and not necessarily to search the slack in all the levels of periodic tasks. Thus, the computational overhead to calculate slack can be reduced.

The remainder of this paper is organized as follows. Section 2 establishes a system model and assumptions used in the paper. Section 3 describes the basic concept of the deadlinewise preassignment scheduling policy. Section 4 describes the optimistic slack calculation approach, explains operations of the APS algorithm, demonstrates the optimality, and gives an example of the algorithm. Section 5 shows the simulation results of the proposed algorithm and compares it with the EDF-CTI algorithm [8]. Finally, we conclude the paper in section 6.

2. Task Model and Assumptions

Consider a uniprocessor real-time system with a set T of *n* independent, preemptable periodic tasks, ($\tau_1, \tau_2, ..., \tau_n$). Each task, τ_i, has a worst-case computation requirement C_i, a period T_i, an initiation time or an offset ϕ_i relative to some time origin, and a deadline d_i. Hence, we denote the system $\mathbf{T} = \{\tau_i(C_i, T_i, d_i): 1 \leq i \leq n\}$. In response to external events which occur at random time instants, the aperiodic tasks, $\{J_k, k \geq 1\}$ are introduced. Each aperiodic request $J_i(a_i, c_i)$ is characterized by its arrival time a_i and its worst-case computation time c_i. Let hyperperiod, ***H***, be the least common multiple of all the periodic task's periods. For simplicity, the algorithm uses the following assumptions.

(A1) Deadline for a periodic task's instance is equal to the next request of the task.

(A2) Preemption over a periodic or an aperiodic task is always possible

(A3) All overhead for context switching is counted into the corresponding periodic or aperiodic task's computation requirements.

(A4) All initiation times or relative offsets $\{\phi_i, 1 \leq i \leq n\}$ are synchronized to 0.

3. The Deadlinewise Preassignment

In this section, we explain the basic concept of the deadlinewise preassignment policy which is the off-line scheduling of the CTI algorithm. For simplicity, we describe the off-line scheduling of the CTI algorithm in

Authorized licensed use limited to: SAMSUNG ELECTRONICS DMC. Downloaded on March 16,2010 at 10:14:57 EDT from IEEE Xplore. Restrictions apply.

the fixed priority systems. The deadlinewise preassignment for a periodic task set in a hyperperiod produces a scheduling table, called the CTI table, which is used in the on-line assignment. Note that the fixed priority based deadlinewise CTI scheduling concept can be easily extended to a dynamic priority based scheduling.

A periodic task scheduling method is classified to a fixed priority ***deadlinewise preassignment*** if the tasks are assigned one after another according to the given fixed priority in such a way that all the tasks are preassigned toward deadlines at their maximum. The priority preemptions in a deadlinewise preassignment take place in a similar manner to the other fixed priority scheduling methods (e.g. rate monotonic priority assignment) except that the part or all of the preempted task instance should be assigned prior to the preempting task instance.

Example 1. Suppose that a periodic task set with two tasks, τ_1 and τ_2, having the computation requirements, C_1=1 and C_2=2, and the periods, T_1=3 and T_2=5, respectively, is to be preassigned over a single hyperperiod, H=15, using the rate monotonic fixed-priority deadlinewise preassignment. The preassigning process is depicted in Figure 1. At start time, two task instances, τ_{11} and τ_{21}, are arrived and assigned toward the deadlines, D_{11}=3 and D_{21}=5, respectively. At time 6, the task instance τ_{13} preempts τ_{22}. Also, at time 12, another preemption is occurred.

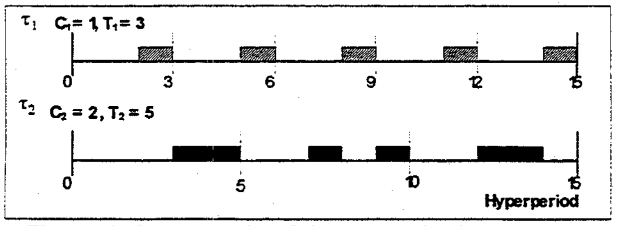

Figure 1. An example of the preassigning process using the deadlinewise preassignment.

Next we establish some of the basic notions used to analyze the feasibility of the deadlinewise preassignment for a given periodic task set. For a set of periodic tasks preassigned according to the deadlinewise preassignment, we say that an ***underflow*** occurs at ***t*** if a task is forced out of its given period beginning at ***t*** as a result of the others' preemptions. The concept of overflow is applicable to both the on-line and the off-line fixed-priority schedulings, but on the other hand that of underflow only to the off-line fixed-priority schedulings. A ***deadlinewise critical instant*** for a given periodic task preassigned according to the deadlinewise preassignment is an instant at which the execution of a request for that task will begin, so that the largest time interval is required to its completion. The periodic interval for a periodic task having the shortest waiting time from the request to the start of the execution contains its deadlinewise critical instant. A ***deadlinewise critical zone*** for a given periodic task preassigned according to the deadlinewise preassignment is the time interval between a deadlinewise critical instant and the deadline for the corresponding request. The above two definitions are hinted at the definitions of the normal critical instant and critical time zone presented in [13]. The only difference between the normal and the deadlinewise is that the latter counts on the deadlines instead of the requests. Any periodic task set with a fixed-priority order is ***deadlinewisely preassignable*** if no underflow occurs through all the deadlinewise critical zones for all the tasks over a single hyperperiod.

Example 2. Figure 1 shows two deadlinewise critical instants, execution start points of τ_{22} and τ_{23}, at priority level 2 and consequently two deadlinewise critical zones, **[7,10]** and **[12,15]**, respectively. Moreover, the periodic task set is deadlinewisely preassignable because no underflow occurs through all the deadlinewise critical zones. Note that every execution start point of the task instances at the priority level 1 is a deadlinewise critical instant.

Meanwhile the property of the deadlinewise preassignment scheduling scheme in the fixed priority systems can be directly implied to the dynamic priority systems.

4. Alternative Priority Scheduling

4.1 Slack Calculation

The major motivation of the proposed algorithm lies how to calculate the slack in a simple manner. In the conventional joint schedulings of aperiodic and periodic tasks, the scheduling time interval **[0, *t*]** is filled up by the execution time of periodic task instances P_{ij} (t), the execution time for aperiodic requests $A(t)$, and the idle time $I(t)$. The formula (1) depicts this concept.

Authorized licensed use limited to: SAMSUNG ELECTRONICS DMC. Downloaded on March 16,2010 at 10:14:57 EDT from IEEE Xplore. Restrictions apply.

$$t = \sum_{i,j} P_{ij}(t) + A(t) + I(t) \quad \cdots \qquad (1)$$

The slack within $[0, t]$, denoted by $S_0(t)$, is generally defined as an available time for the aperiodic requests while guaranteeing the deadlines of the given periodic requests. Thus, $S_0(t)$ is a total amount of time from 0 to t except for the sum of the execution time of periodic requests.

The actual slack for aperiodic requests, however, is less than or equal to $S_0(t)$ because it includes idle time (see formula (2)).

$$S_0(t) = A_0(t) + I_0(t) \quad \cdots \qquad (2)$$

The slack stealing algorithm [10],[11] uses the actual slack as the minimum slack value of all periodic requests to ensure that all the lower priority periodic deadlines are met (see formula (3)).

$$S^*(t) = min_{\{1 \le i \le n\}} S_i(t) \quad \cdots \qquad (3)$$

On the contrary, in the APS algorithm, we have newly introduced the concept of an optimistic slack calculation method to get a maximum slack $S(t)$ at current time t. The idea of the optimistic approach is that a scheduler regards whole of the time as a slack except for the execution time of the given periodic tasks to meet their deadlines. Thus, the optimistic approach is not necessarily to calculate slack at all the levels of periodic tasks which in turn reduces the computational complexity to calculate slack.

$$S(t_1, t_2) = t_2 - t_1 - \sum_{i=1}^{n} \delta_i \quad \cdots \qquad (4)$$

The formula (4), which shows a way of an optimistic slack calculation, demonstrates an actual value of slacks in $[t_1, t_2]$. $S(t_1, t_2)$ can be obtained by subtracting the minimum amount of the execution time of periodic tasks ($\sum_{i=1}^{n} \delta_i$) from $(t_2 - t_1)$.

In order to get slack efficiently, we can decompose a hyperperiod into a set of scheduling zones, denoted by $[t, Z(t)]$ where $Z(t)$ can be calculated by the formula (5). The meaning of $[t, Z(t)]$ is a time interval from t to the earliest deadline of a periodic task. The reason for introducing the decomposition strategy is to perform an efficient slack calculation by using $Z(t)$ as a scheduling checkpoint. For example, when an aperiodic request happens at any arbitrary time t, the foremost concern is how to schedule that aperiodic task while guaranteeing the very following deadline of periodic task's instance. By this way, we can easily evaluate the slack availability while considering the deadlines of periodic tasks. Notice that $Z(t)$ can be obtained from the sequences of periodic tasks' instances in the CTI table.

$$Z(t) = min\{d_{ij} | 1 \le i \le n, j = \left\lceil \frac{t}{T_i} \right\rceil\} \cdots \qquad (5)$$

Consequently, $Z(t)$ is a parameter to calculate slack from current time t to the earliest deadline of the periodic task. Thus, we can calculate the slack as follows:

$$S(t) = S(t, Z(t)) = Z(t) - t - \sum_{i=1}^{n} \delta_i \cdots \qquad (6)$$

In the formula (6) which can be easily induced from the formula (4), we call δ_i as the ***cet*** and it can be obtained from the CTI table. Recall that the ***cet*** is used to evaluate whether the preassigned periodic tasks in the CTI table should be executed or not to meet their deadlines. To clarify the term ***cet***, we introduce two notations; one is the cumulation of all the computation processing completed ($Cp_i(t)$) and the other is the cumulation of all the computation requirement ($Cr_i(Z(t))$) in the CTI table. Hence, we can formally define the ***cet*** as follows:

$$\delta_i = \begin{cases} 0, & \text{if } (Cr_i(Z(t)) - Cp_i(t)) < 0 \\ (Cr_i(Z(t)) - Cp_i(t)), & \text{otherwise} \end{cases} \cdots \quad (7)$$

For simple notation, we define the sum of ***cet***s at time t as follow:

$$\Delta(t) = \sum_{i=1}^{n} \delta_i \qquad \cdots \qquad (8)$$

4.2 Operation of the Algorithm

As we mentioned before, the APS algorithm chooses either an EDF or a CEF scheduling alternatively at on-line for a given periodic task set based on $\Delta(t)$ while referencing information in the off-line built CTI table.

The major reason to introduce an alternative scheduling policy is to get fast response time for aperiodic requests. Suppose there is no aperiodic request. At that time, if the scheduler performs on-line scheduling using EDF only, then some periodic tasks (***non-critical tasks***) of which deadlines are earlier than those of the ***critical tasks*** may be scheduled first (in case of a deadline of a non-critical task is earlier than that of a ***critical task***.) In this case, the scheduler finds no more slack available since the value of slack obtained from

Authorized licensed use limited to: SAMSUNG ELECTRONICS DMC. Downloaded on March 16,2010 at 10:14:57 EDT from IEEE Xplore. Restrictions apply.

formula (6) turns to be negative. Note that a ***critical task*** does not mean a periodic task which has the earliest deadline in a given periodic task set. Therefore, as the value of $\Delta(t)$ is increased, in consequence, the value of slack is decreased. Consequently, an aperiodic task may not be scheduled under EDF policy even though the slack is available. On the other hand, in the CEF algorithm, since the scheduler allocates the minimum amount of execution of periodic tasks within $Z(t)$ based on the information in the off-line built CTI table, the slack value obtained from the formula (6) will never be a negative which in turn has higher probability to find more slack.

Recall that we define that the CEF scheduling gives the highest priority to the periodic task which has the largest value of the ***cet***. Thus, if there is periodic task(s) in the ready queue and $\Delta(t) > 0$, then service the periodic task with the CEF scheduling policy. Obviously, we can get the $\Delta(t)$ from information in the CTI table. In other words, if ***cet*** > 0, then the scheduler must follow the information in the CTI table that instructs the scheduling (assignment sequence) of periodic tasks.

```
1   Build off-line CTI table
2   Initialize counters
3   while TRUE do
4     Calculate S(t) and Cri(Z(t));
5     while (t < Z(t)) do
6     if (there is slack and there is aperiodic task in
          the queue) then
7         while (S(t) > 0 and there is aperiodic task in
                 the queue) do
8             Service the aperiodic task;
9             Update S(t)
      /* reduce the amount of service time from S(t) */
10    else if (there is periodic task in the queue) then
11        if (Δ(t)>0) then
12                Service the periodic task with the CEF;
13            Update Cpi(t)
      /* add the amount of processing time */
14        else
15            Service the periodic task with the EDF;
16            Update Cpi(t)
      /* add the amount of processing time */
17            Update S(t)
      /* reduce the amount of processing time */
18    else
19        Process idle state;
20        Update S(t)
      /* reduce the amount of idle time */
21    endwhile
22  endwhile
```

Figure 2. A pseudocode of the APS algorithm

In this subsection, we describe the APS algorithm as well as its operational behavior using Statechart [4]. The following Figure 2 depicts a pseudocode of the algorithm.

At line 1 and 2 in Figure 2, the off-line CTI table is created and the scheduling parameters are initialized. At line 4, the scheduler calculates slack. From line 6 to 9, aperiodic tasks are serviced if slack is available and there are aperiodic tasks in the queue. At line 10 and 11, if there are periodic tasks, the algorithm alternatively chooses either an EDF or a CEF scheduling based on $\Delta(t)$. At line 19, if there is no periodic and aperiodic task to be serviced, the scheduler is idle.

In order to clarify the operational behavior of the APS algorithm, we demonstrate its state transition using the Statechart [4]. In Figure 3, the transition diagram consists of A, I, and P states which stand for aperiodic task service, idle, and periodic task service respectively. Further, P state is divided into an EDF and a CEF state. Note that $Q_p(t)$, $Q_a(t)$, and $S(t)$ denote the sum of the execution time of periodic tasks at time t, the execution time of aperiodic tasks at time t, and the available slack respectively.

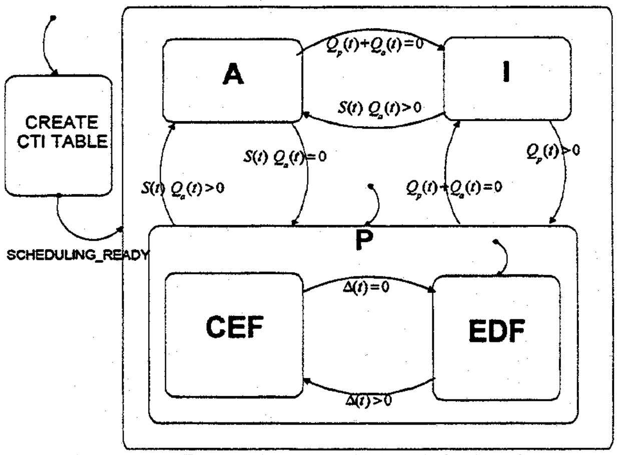

Figure 3. The behavior of the APS algorithm

The description of each state is as follows;

- A state: This is a state of the aperiodic service. If a slack value is greater than 0 and there is an aperiodic task ($S(t).Q_a(t) > 0$), then a transition will occur from P or I to this state. In this state, the slack value will be decreased as much amount as the aperiodic service is done.
- I state: This is an idle state. If there is no periodic or aperiodic task ($Q_p(t)+Q_a(t) = 0$), then a transition

Authorized licensed use limited to: SAMSUNG ELECTRONICS DMC. Downloaded on March 16,2010 at 10:14:57 EDT from IEEE Xplore. Restrictions apply.

will happen from A or P to this state. In this case, the slack value will be decreased as the same amount of the idle time.

- **P** state: This is a state of the periodic service. If there is no aperiodic task or the value of slack is 0 ($S(t).Q_a(t) = 0$), then a transition will be triggered from A state. Also, if a periodic task arrives ($Q_p(t) > 0$), then a transition will occur from I to this state.
- **CEF** state: If $\Delta(t) > 0$, then a transition will occur from the EDF to this state.
- **EDF** state: If $\Delta(t) = 0$, then a transition from the CEF to this state will be occur.

4.3 Optimality of the Algorithm

In order to show optimality of the algorithm in the CTI scheduling framework, we introduce the following theorems.

Theorem 1: At the starting point of the hyperperiod, the EDF deadlinewise preassignment [3] in a dynamic priority system, as an off-line scheduling, is an optimal scheduling for the aperiodic requests.

Proof. At the start time of the hyperperiod, the deadlinewise preassignment is a special case of the EDL since the EDL has the property that assign the periodic tasks as late as possible. The EDL, meanwhile, is known to be an optimal aperiodic scheduling in the EDF framework which was proven by Chetto and Chetto [1]. ■

Theorem 2: In the CTI scheduling framework, the APS algorithm can find maximum slack at any arbitrary time t during on-line scheduling.

Proof. Suppose there is an algorithm *X* which can find more slack than that found by the APS algorithm in [*t*, *Z*(*t*)]. Also, suppose all of the slack found by the algorithm *X* was consumed for the aperiodic tasks. Then sometimes later than *Z(t)*, the algorithm *X*, in worst case, may not guarantee the deadlines of periodic tasks because it can not consider the executions and deadlines of upcoming periodic tasks to be scheduled after *Z(t)*. In other words, since the *cet* is preserved as the minimum time to guarantee the deadline of periodic tasks to *Z(t)*, the algorithm *X*, which found more slack than APS, is consequently can not guarantee the deadlines of periodic tasks after *Z(t)*. Further, if a certain greedy scheduling algorithm in dynamic priority system can find a maximum slack, then it is an optimal algorithm in term of fast response time of aperiodic requests. Thus, the APS algorithm is optimal within the CTI scheduling framework which can find the maximum slack at arbitrary time *t*. ■

4.4 An Example

In this subsection, we demonstrate an example of scheduling using the APS algorithm. Suppose there are three periodic tasks $\tau_1(1,5,5)$, $\tau_2(1,7,7)$, $\tau_3(3,10,10)$ where the parameters are the worst case computation time, deadline, and period of the tasks, respectively. We assume that the first aperiodic task J_1 arrives at $a_1=4$ with its computation time $c_1=3$. Then, we can calculate the slack S(0) as follow:

$$S(0) = S(0, Z(0)) = S(0,5) = 5 - 0 - \sum_{i=1}^{3} \delta_i$$
$$= 5 - (1+0+0)$$
$$= 4$$

Thus, the available slack within **[0, 5]** becomes 4 units. Since there is no aperiodic task at t=0, the critical task τ_{11} is invoked by the CEF algorithm. From t=1 to t=4, τ_{21} and τ_{31} are invoked by the EDF algorithm and 3 units of the slack value are consumed which means S(4)=1. At time t=4, the aperiodic task J_1 arrives. Since one unit of slack is available by t=5, J_1 is serviced immediately. At t=5, S(5) can be obtained as follows:

$$S(5) = S(5, Z(5)) = S(5,10) = 10 - 5 - \sum_{i=1}^{3} \delta_i$$
$$= 5 - (1+0+1)$$
$$= 3$$

Since τ_{21} is already completed its execution before t=5, the next earliest absolute deadline is t=10. Thus, Z(5)=10 instead of 7. Meanwhile, because S(5)=3, the remaining 2 units of aperiodic task's execution time will be invoked.

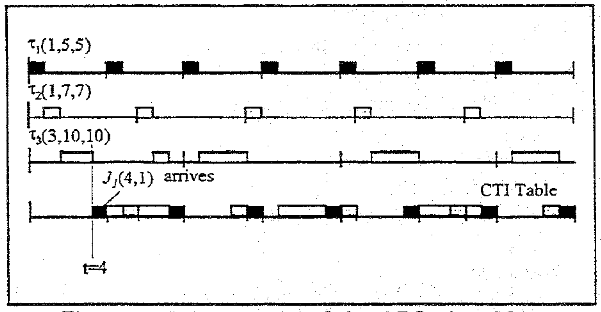

Figure 4. An example of the APS algorithm

Authorized licensed use limited to: SAMSUNG ELECTRONICS DMC. Downloaded on March 16,2010 at 10:14:57 EDT from IEEE Xplore. Restrictions apply.

5. Simulation

In this section, we observe the average response time for aperiodic requests. The performance of the APS scheduling will be compared to that of all the combinations of a fixed- or dynamic-CTI table built off-line and an on-line fixed- or dynamic-priority assignment. Thus, four different types of combinations are available.

The task set in the simulation consists of 10 different periodic tasks, each of which has randomly generated period and computation requirements. All aperiodic tasks are generated by using both an exponential distribution function for their computation requirements and the Poisson arrival function for their arrivals. An aperiodic workload has been coordinated by modifying the value of the exponential scale parameter and the arrival rate of the Poisson function. We arranged the periodic task set with 90% of CPU utilization to simulate high workload scheduling. It is summarized in Table 1. Note that the average execution time for aperiodic tasks is set to 3.5 and 21.

	With 90% Periodic Workload	
Task ID	Period	Computation
1	100	2
2	280	14
3	2100	108
4	440	29
5	350	14
6	210	30
7	35	8
8	70	11
9	2200	231
10	300	12

Table 1. A sample periodic task set.

In Figure 5, as the workload of aperiodic tasks is increased, the average aperiodic response time tends to be slower. The average response time is dependent more on the off-line scheduling if the aperiodic workload is low, but is dependent more on the on-line scheduling if the aperiodic workload is high. Figure 6 shows the result in case that the average execution time is set to 21. In this case, we observe that the average response time for soft-aperiodic tasks is dependent more on the average execution time; the larger average execution time for aperiodic tasks, the slower average response time for that tasks. The simulation study shows that the average response time for aperiodic tasks in the proposed algorithm is, in most cases, slightly better than that of the EDF-CTI.

Moreover, the simulation result shows that the APS algorithm has slightly faster response time than the EDF-CTI in a high workload (over 97%). Note that the computational complexity of the APS off-line approach is $O(n\log_2 n)$ [12] while that of on-line is $O(n)$ where n is the number of periodic tasks.

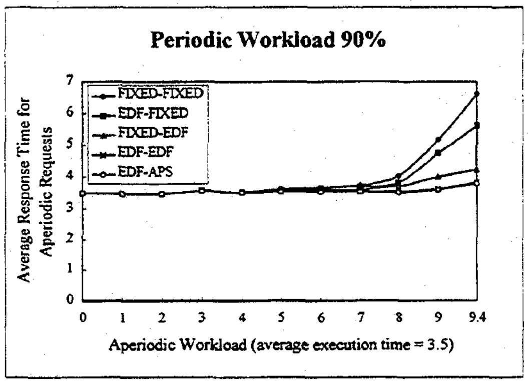

Figure 5. The simulation result in case that the average execution time of aperiodic requests is 3.5

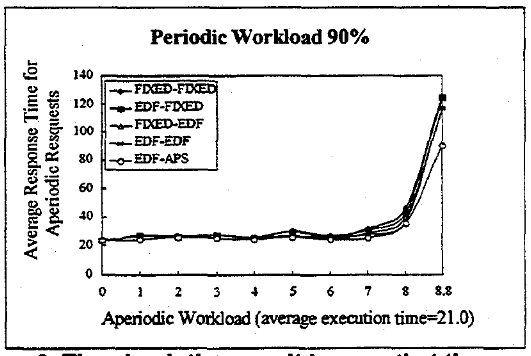

Figure 6. The simulation result in case that the average execution time of aperiodic requests is 21

6. Summary

The slack stealing based schedulings require the computationally expensive re-evaluation of the slack at each priority level. In this paper, we introduced an efficient method of calculating the slack. The APS algorithm performs on-line scheduling using either an EDF or a CEF scheduling policy alternatively to guarantee all the deadlines of the periodic tasks while referencing the information in the off-line built CTI table. In order to reduce the computational overhead of slack

Authorized licensed use limited to: SAMSUNG ELECTRONICS DMC. Downloaded on March 16,2010 at 10:14:57 EDT from IEEE Xplore. Restrictions apply.

calculation, we newly adopted a concept of an optimistic approach to maximize reclaiming of unused computation time. Therefore, the algorithm is not necessarily searching the slack in all the levels of periodic tasks, which turns out be more practical. We also demonstrated that the proposed algorithm is optimal in terms of calculating a maximum slack when aperiodic request has occurs at certain time *t*. Our simulation study shows that the APS algorithm, in most cases, is slightly better than the EDF-CTI algorithm which is even superior to the other CTI-based soft-aperiodic schedulings in terms of the short response time of aperiodic requests.

When developing the scheduling algorithms, we should consider the practical applications in real-world. Our algorithm, in this respect, does satisfy the practicability and simplicity. By using the alternative scheduling policy, we can get a fast average response time for aperiodic requests. By introducing the ***optimistic*** slack calculation method, we can achieve the implementation simplicity. Our ongoing work is to enhance the algorithm to be more robust in the transient overload and to compare our simulation results to that of Tia's [19].

References

[1] Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, Vol. 15, No. 10, pp. 466-473, 1989.

[2] Davis and A. Wellings, "Dual Priority Scheduling", Proceedings of the IEEE Real-Time Systems Symposium, pp. 100-109, December 1995.

[3] Davis, K. Tindell and A Burns, "Scheduling Slack Time in Fixed Priority Pre-emptive Systems", Proceedings of the IEEE Real-Time Systems Symposium, pp. 222-231, December 1993.

[4] Harel, "Statecharts: A Visual Formalism for Complex Systems," Science of Computer Programming, Vol. 8, pp. 231-274, 1987

[5] Homayoun and P. Ramanathan, "Dynamic Priority Scheduling of Periodic and Aperiodic Tasks in Hard Real-Time Systems", Real-Time Systems: The International Journal of Time-Critical Computing Systems, Vol. 6, No. 2, pp. 207-232, 1994

[6] Kim, S.Y. Lee and J.W. Lee, "A Near-Optimal Algorithm for Scheduling Soft-Aperiodic Requests in Dynamic Priority Systems", The 8th Euromicro Workshop on Real-Time Systems, June, 1996, to appear.

[7] Lee, S.Y. Lee, and H.I Kim, "Scheduling Hard-Aperiodic Tasks in Hybrid Static/Dynamic Priority Systems", ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems, pp. 7-19, June, 1995.

[8] Lee, H.I. Kim and J.W. Lee, "A Soft Aperiodic Task Scheduling Algorithm in Dynamic Priority Systems", 2nd IEEE Workshop on Real-Time Computing Systems and Applications, Tokyo, pp. 68-72, October, 1995.

[9] Lehoczky, L. Sha, and J.K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", Proceedings of the IEEE Real-Time Systems Symposium, pp. 261-270, San Jose, CA, December 1987.

[10] Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems", Proceedings of the IEEE Real-Time Systems Symposium, pp. 110-123, December 1992.

[11] Lehoczky and S. R. Thuel, Scheduling Periodic and Aperiodic Tasks using the Slack Stealing Algorithm (Chapter 8), Advances in Real-Time Systems, (ed. S. Son) Prentice-Hall, Englewood Cliffs, NJ, 1995.

[12] Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks", Performance Evaluation 2, pp. 237-250, 1982.

[13] Liu and J.W. Layland, "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environments", Journal of the Association for Computing Machinery, Vol. 20, No.1, pp. 46-61, January 1973.

[14] Shin and Y.-C. Chang, "A Reservation-Based Algorithm for Scheduling Both Periodic and Aperiodic Real-Time Tasks", IEEE Transactions on Computers, Vol. 44, No.12, pp. 1405-1419, December, 1995.

[15] Spuri and G.C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling", Proceedings of the IEEE Real-Time System Symposium, pp. 2-11, December, 1994.

[16] Spuri, G.C. Buttazzo, and F. Sensini "Robust Aperiodic Scheduling Under Dynamic Priority Systems", Proceedings of the IEEE Real-Time System Symposium, pp. 210-219, December, 1995.

[17] Sprunt, J.P. Lehoczky, and L. Sha, "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System", Technical Report CMU/SEI-890TR-11, April, 1989.

[18] Sprunt, J.P. Lehoczky, and L. Sha, "Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm", Proceedings of the IEEE Real-Time System Symposium, pp. 251-258, December, 1988.

[19] Tia, Utilizing Slack Time for Aperiodic and Sporadic Requests Scheduling in Real-Time Systems, Technical Report No. UIUCDCS-R-95-1906, University of Illinois, April, 1995.

Authorized licensed use limited to: SAMSUNG ELECTRONICS DMC. Downloaded on March 16,2010 at 10:14:57 EDT from IEEE Xplore. Restrictions apply.