A Near-Optimal Algorithm for Scheduling Soft-Aperiodic Requests in Dynamic Priority Systems

Hyungill Kim[9], Sungyoung Lee[9], Jongwon Lee[99], and Dougyoung Suh[999]

Department of Computer Engineering, Kyung Hee University, Seoul, Korea[9]
Software Research Lab., Korea Telecom, Seoul, Korea[99]
Department of Electronics, Kyung Hee University, Seoul, Korea[999]

ABSTRACT

In this paper, we propose a reasonably simple and near-optimal soft aperiodic task scheduling algorithm in dynamic priority systems. The proposed algorithm has extended the EDF-CTI (Earliest Deadline First-Critical Task Indicating) Algorithm [4] in such a way of modifying the slack calculation method which in turns resolves the unit scheduling and the critical task misindicating problems. The paper demonstrates a simple way of slack calculation and near optimality of the algorithm. Our simulation study shows that the proposed algorithm, in most case, is slightly better than the EDF-CTI algorithm in terms of short response time of aperiodic requests, and considerably improves that in a high workload.

1. Introduction

In the last few decades considerable researches have been done in the area of soft aperiodic task scheduling in fixed priority systems [8],[9] such as the polling server, the bandwidth preservation [5],[11],[12], and the slack stealing algorithm [6],[7]. Recently, Tia and Liu [13] introduced a scheduling of aperiodic requests in dynamic priority systems [9] 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 are 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 optimal in that it minimizes the response times of the aperiodic requests. Homayoun and Ramanathan [2] 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 the aperiodic requests are sometimes not the minimum possible. Spuri and Buttazzo [10] 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, however, its run time overhead is higher than that of Tia and Liu's algorithm.

In this paper, we introduce a new type of near-optimal soft aperiodic task scheduling, called dynamic CTI algorithm, in dynamic priority systems. Some optimal schedulings for soft aperiodic requests in dynamic priority systems already exist, so a natural question is: why near-optimal? The answer is that none of the existing optimal soft aperiodic task schedulings satisfy the requirements of practicality in real-world applications due to their computational overhead for slack calculation. Our goal, consequently, is to find a near-optimal scheduling for soft aperiodic requests, which is still preserving less computational overhead than the other existing algorithms. To achieve this, we have adopted a scheduling scheme in which the EDF scheduling is applied to the given periodic task set while referencing the information on the off-line built dynamic CTI (Critical Task Indicating) table [3]. The dynamic CTI table is built by applying the deadlinewise preassignment scheme to the given periodic task set in a dynamic priority system. The scheduling information on this static table enables us to have a scheduling

predictability and reduces the overall computational complexity at run time. Building a CTI table is very similar to the reverse schedule of the EDL introduced by Chetto and Chetto [1]. However, there are differences between two approaches in terms of their applications and pursuing goal which are shown in the [4].

The remainder of this paper organized as follows. Section 2 establishes a system model used in the paper. In section 3, we introduce the dynamic CTI algorithm and addresses its near optimality. Section 4 demonstrates the simulation result of the proposed algorithm and compares it with the EDF-CTI algorithm [4]. Finally, we conclude this paper in section 5.

2. Task Model

Consider a uniprocessor real-time system with a set **T** of n independent, preemptable periodic tasks, (τ_1, τ_2, ..., τ_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 $\mathbf{T} = \{\tau_i(C_i,T_i,d_i): 1 \le i \le n\}$. We assume that $T_i = d_i$ and all initiation times or relative offsets $\{\phi_i, 1 \le i \le n\}$ are synchronized as 0. In response to external events which occur at random time instants, the aperiodic tasks, $\{J_k, k \ge 1\}$ 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.

3. Dynamic CTI Algorithm

In the conventional joint scheduling 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.

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

Meanwhile, the slack within [0,t], denoted by $S_0(t)$, is generally defined as the available time for aperiodic requests while guaranteeing the deadlines of the given periodic requests. Consequently, $S_0(t)$ is the 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)).

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

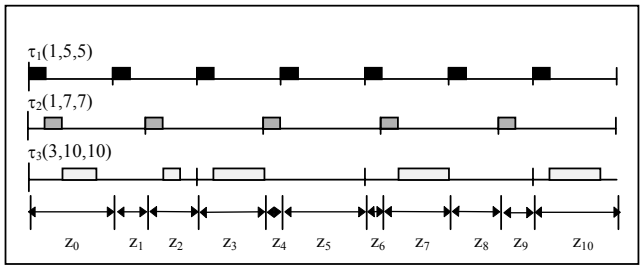
The slack stealing algorithm [6],[7] uses the actual slack as the minimum slack value to ensure that all the lower priority periodic deadlines are met (see formula (3)).

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

On the contrary, in our proposed algorithm, the maximum slack $S(t)$ at the current time t, can be obtained by the formula (4) where D is the absolute deadline of a periodic task.

$$S(t) = D - t - \sum_{i=1, d_i \in [t,D]}^{n} C_i \qquad (4)$$

In order to get $S(t)$ efficiently, we decompose a hyperperiod into a set of scheduling zones, denoted by $\{Z_k, 0 \le k\}$ where k is an index. The actual domain of the decomposition is a virtual scheduling table produced by the EDF on-line scheduling. A Z_k is a time interval between two deadlines of instances of periodic tasks. The main reason for introducing the decomposition mechanism is to reduce the computational complexity for slack calculation and to find the available slack as much as possible. By this way, we can easily evaluate the slack availability while considering the deadlines of periodic tasks. Figure 1 depicts an example of how to obtain $\{Z_k\}$ for the given 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.



[Figure 1] An example of $\{Z_k\}$ decomposition.

In order to explain the algorithm and to show it is near-optimal, we introduce the following fundamental theorems.

Theorem 1: At the starting point of the hyperperiod, the deadlinewise preassignment scheme [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]. ■

In the algorithm, the most important factor is to decide whether the given periodic tasks should be executed or not. We, therefore define that $\mathbf{Cr_i}$ is the computation requirement for τ_i which should be completed and $\mathbf{Cp_i}$ is all the computation processing done for τ_i.

Theorem 2: Let $P_k(t)$ be the execution time of periodic tasks which is not preassigned in the CTI table section that correspond to the on-line scheduling zone Z_k. The lost slack as much as the amount of $P_k(t)$ within Z_k during the on-line scheduling will be compensated at the next adjacent scheduling zone Z_{k+1} and/or later zones.

Proof: If there are no critical tasks, any periodic task which has the shortest deadline should be assigned based on the EDF. In this case, the amount of the slack loss caused by the processing of the periodic tasks will be compensated at the next adjacent scheduling zone. That is, the amount of lost slack will be represented as follows :

$$P_k(t) = \sum_{d_{i,j} \in Z_{k+1}} (Cp_i(t) - Cr_i(t)) + \cdots + \sum_{d_{i,j} \in Z_{k+l}} (Cp_i(t) - Cr_i(t)) \qquad (5)$$

where $P_k(t) \le S_k$, $1 \le l < N$.

In this formula, S_k denotes the maximum slack value produced by the off-line CTI table section corresponding to Z_k and N denotes the number of periodic requests in the hyperperiod.

From the formula (5), we can induce a new formula (6) that calculates the maximum slack in Z_k.

$$S_k(t) = S_k + \sum_{d_{i,j} \in Z_k} (Cp_i(t) - Cr_i(t)) \ldots (a) - P_k(t) - I_k(t) - A_k(t) \ldots (b) \qquad (6)$$

In the formula (6), **(a)** is the compensated slack value from the previous adjacent scheduling zone and **(b)** is the total lost slack value by the current time t in Z_k such as the execution time of periodic tasks, the idle time, and the execution time for aperiodic requests.

The formula (5) and (6) prove the theorem. ■

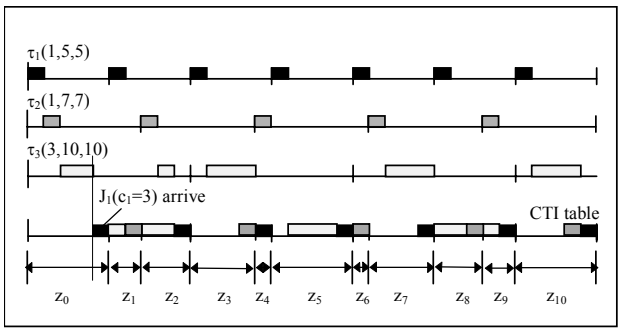
From the Theorem 1 and 2, it is shown that the slack is retained at the maximum within Z_k. This fact supports that the algorithm keeps the maximum slack within $[0,t]$ since the interval $[0,t]$ is composed of the discrete consecutive scheduling zones. Hence, the proposed algorithm is near-optimal for soft aperiodic requests at any time t. Figure 2 shows a pseudocode of the dynamic CTI algorithm.

Example. In Figure 3, we demonstrate an example of scheduling for the proposed algorithm. We assume that the first aperiodic task arrives at t = 4 with its computation time c_1=3. **(a)** When the aperiodic task

```
Build CTI table at off-line
Initialize various counters including Cp_i, Cr_i, etc.
if (there is aperiodic task in the queue) then
   Determine S_k(t), the maximum slack available within Z_k at the current time t;
    if (there is slack) then
        while (S_k(t) > 0 and there is aperiodic task in the queue) do
            Service the aperiodic task;
            Update A_k, the accumulation of elapsed processing time for aperiodic tasks within Z_k
    else
        Service the periodic task with the highest priority;
        Update Cp_i, the accumulation of elapsed processing time for the periodic task from the start of
            the hyperperiod
else
    if (there is periodic task in the queue) then
        Service the periodic task with the highest priority;
        Update Cp_i
    else
        Process idle state;
        Update I_k, the accumulation of elapsed idle processing time within Z_k
```

[Figure 2] A Pseudo Code of the Dynamic CTI Algorithm

arrived, S_0=0, which calculated from the CTI table section corresponding to Z_0. Since the tasks τ_{21} and τ_{31}, which are not preassigned to the CTI table section that correspond to Z_0, have been executed at t=1 and t=2 respectively, $P_0(4)$=3, $A_0(4)$=0, and $I_0(4)$=0. Since Z_0 is the first scheduling zone in the hyperperiod, the compensated slack value in the next adjacent scheduling zones is equal to 0. Consequently, because $S_0(4)$=1, we can assign the slack value 1 to the aperiodic request within that zone. **(b)** At time 5, the starting time of the next scheduling zone Z_1, all the values of S_1, $P_1(5)$, $I_1(5)$, and $A_1(5)$ are equal to 0. However, since the reserved slack value within Z_0 is equal to 2, $S_1(5)$=2. Therefore, J_1 is to start its execution at t=4 and to complete at t=7.



[Figure 3] A scheduling example

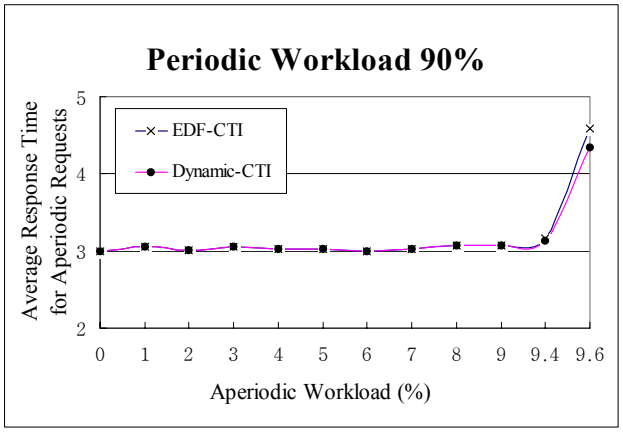
4. Simulation Result

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 easily coordinated by modifying the exponential scale parameter value 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.

	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.



[Figure 4] Simulation result

In Figure 4, 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. 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. Especially, the algorithm shows considerably fast response time compared to the EDF-CTI in a high workload (over 97%). Meanwhile, the computational complexity of our off-line approach is $O(\boldsymbol{n}\log_2 \boldsymbol{n})$ while that of on-line is $O(\boldsymbol{n})$ where $\boldsymbol{n}$ is the number of periodic tasks.

5. Summary

We introduced a near-optimal scheduling for soft aperiodic requests in dynamic priority systems. The proposed algorithm has simplified the way of slack calculation in such a way of partitioning the hyperperiod into the scheduling zones which resolves the problems of the EDF-CTI algorithm such as the unit time scheduling and the critical task misindicating. By this method, the algorithm considerably reduces the computational complexity which enables the algorithm to be more practical. Without saying, every scheduling algorithm should be developed for the practical real-world applications. In this respect, our algorithm meet the claim, the implementation simplicity and the fast average response time for aperiodic requests. Our ongoing work is to develop a formal theory on the optimality of the algorithm that makes it more concrete and robust.

Reference

[1] H. 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] N. 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

[3] J.W. 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, La Jolla, CA, June, 1995.

[4] S.Y. 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.

[5] J.P. 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.

[6] J.P. 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.

[7] J.P. 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.

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

[9] C.L. 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.

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

[11] B. 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.

[12] B. 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.

[13] T.S. 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.