A Soft Aperiodic Task Scheduling Algorithm in Dynamic-Priority Systems

Sungyoung Lee[J], Hyungill Kim[J], Jongwon Lee[JJ]

[J]Department of Computer Engineering
Kyung Hee University, Seoul, Korea
[JJ]Software Research Lab.
Korea Telecom, Seoul, Korea

Abstract

In this paper, we present a joint scheduling of hard deadline periodic and soft deadline aperiodic tasks in dynamic-priority systems. The proposed algorithm has extended the fixed-Critical Task Indicating (CTI) algorithm developed by the authors [3]. The dynamic-CTI algorithm is operated in such a way that dynamic-priority assignment strategy for a given periodic task set and the information on a deadlinewise preassignment table, called dynamic-CTI table, are mixed dynamically according to the aperiodic tasks' arrivals at runtime. The algorithm has a predictability and a less computational complexity in calculating the slacks since it uses the dynamic-CTI table. Our simulation study shows that the dynamic-CTI scheduling has better performance than the fixed-CTI algorithm which is even better than the slack stealing algorithms, especially under a heavy transient overload.

1. Introduction

In the last several years, the joint scheduling algorithms of periodic and soft aperiodic tasks in fixed priority systems [8],[9] have been investigated by many researchers [5],[6],[11],[12]. However, relatively a few works have been done in dynamic priority systems. Recently, Homayoun and Ramanathan [2] extended the deferrable server scheduling algorithm [5] 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. Tia and Liu [13] developed a scheduling of aperiodic requests with a slack-stealing approach. The algorithm, which is a greedy, uses the EDF algorithm to schedule the periodic requests. The time complexity of the algorithm is O(n) where n is the number of periodic tasks. It is optimal in that it minimizes the response times of all the aperiodic requests among all valid aperiodic task scheduling algorithms. Spuri and Buttazzo [10] also introduced an optimal algorithm, the EDL algorithm. However, its runtime overhead is higher than that of Tia and Liu's algorithm.
In this paper we introduce a different type of the soft aperiodic task scheduling algorithm in dynamic-priority systems. The proposed algorithm extended the fixed-CTI algorithm [3] which is a soft aperiodic task scheduling in fixed priority systems. The simulation study [3] showed that the fixed-CTI algorithm has a faster response time for aperiodic requests and a less computational complexity in calculating slacks than the slack stealing algorithms [6],[7]. The major goal of the dynamic-CTI algorithm is to provide a scheduling predictability and a simplicity in calculating slacks. To achieve this, we have adopted a scheduling mechanism in which a dynamic-priority assignment strategy for a given periodic task set and the information on a off-line built dynamic-CTI table are mixed according to the aperiodic tasks' arrivals at run-time. The role of the dynamic-CTI table is to indicate the critical periodic tasks (if any) at each scheduling point that must be assigned immediately to meet their deadlines. This property enables us to have a scheduling predictability and reduces the overall computational complexity at run-time since it has scheduling information in a priori.

To build a dynamic-CTI table at off-line, we developed the dynamic-priority preassignment strategy, called deadlinewise preassignment, in which each start time of the given set of periodic tasks' instances is deferred toward their deadlines at maximum according to the given dynamic-priority. Consequently, the slacks which were made by artificially deferring the execution time of periodic tasks can be utilized by the aperiodic tasks if they arrive at those slack zones. Otherwise, those slacks can be used for the periodic tasks just as per normal dynamic-priority scheduling. The property of the deadlinewise preassignment is shown on [3], [4] in detail.

Intuitively, the deadlinewise preassignment for a periodic task set can also be obtained by rotating the dynamic-priority preassignment in a 180° on the axis of the beginning or the ending point of the hyperperiod under the strong assumption that the deadlines of periodic tasks must be the same as the periods of them. Because of this transposed property for a special case, the CTI approach may be considered as same method as the reverse schedule, introduced by Chetto and Chetto [1]. We, however, would like to point out the major differences between the Chetto and Chetto's work [1] and ours in terms of the application scope (constraints) and the pursuing goal. In the light of the application scope, our approach can be used to relax the constraint of Chetto and Chetto's scheduling method. While the reverse schedule of EDL (*Earliest Deadline as Late as possible*) proposed by them is restricted to the fact that the deadlines of periodic tasks must be same as the periods of periodic tasks, an extension of our approach will

remove the limitations by building a dynamic-CTI table using deadlinewise preassignment according to the EDF priority order. By using this deadlinewise preassignment, the dynamic-CTI approach does not need to reverse the scheduling domain, so that the deadlines of the periodic tasks are not necessarily to be same as the periods of those periodic tasks. Further, Chetto and Chetto have introduced the EDL scheme as an acceptance test mechanism for hard aperiodic tasks while we have suggested the CTI approach as a mechanism for mainly reducing computational complexity in calculating the slacks.

The remainder of the paper is organized as follows. Section 2 describes the dynamic-CTI algorithm including an example. Section 3 shows simulation results and addresses some issues on the CTI approach. Section 4 discusses an extension of the proposed algorithm. Finally, we conclude the paper in section 5.

2. The Dynamic-CTI Algorithm

The dynamic-CTI algorithm seems to be similar to the Chetto and Chetto's approach in that EDL scheduling method at off-line is mixed with EDS (*Earliest Deadline as Soon as possible)* scheduling at on-line. We, however, already mentioned the differences between the two approaches in section 1.

A dynamic-CTI table is obtained by applying the deadlinewise preassignment strategy based on a dynamic-priority policy for the given instances of periodic tasks. This approach is very similar to creating the fixed-CTI table. The dynamic-CTI algorithm also works on the same principle as of the fixed-CTI algorithm. Figure 1 depicts a pseudocode of the dynamic-CTI algorithm.

```
1 initialize data structures
   /* create the dynamic-CTI table */
2 loop begin
3 if (a critical periodic task unit not yet been serviced
      has occurred)   then service it
4    else if (aperiodic task(s) is ready or arrived) then
            service it
5    else if (periodic task(s) is ready or arrived) then
            service it
6    else process CPU idle state
7  advance timer
8  if ((timer_value MOD hyperperiod) is equal to zero)
            then reinitialize the global parameters
9 end loop
```

Figure 1. A pseudocode of the dynamic-CTI algorithm.

At line 1, the data structures for the algorithm including the dynamic-CTI table, the timer, and the hyperperiod are initialized for a given periodic task set. At line 3, the algorithm checks to see if ***(C1) there are slacks available for the aperiodic task in the CTI table at the current time, or (C2) the current critical task unit (whole or part of the task instance) already has been serviced.*** If one of the two conditions is met, it services the aperiodic task immediately. If neither of them is met, the critical task is serviced. Otherwise, an aperiodic task in a ready state (or newly arrived) is serviced immediately at line 4. A dynamic scheduling such as the EDF algorithm is to be applied to the periodic tasks at line 5. Note that the assumptions for the dynamic-CTI algorithm are basically same as that for the fixed-CTI scheduling.

Example 1. Suppose there is a set of periodic tasks T_s which consists of three periodic tasks. $T_s = \{\tau_1(1,5,5), \tau_2(5,10,10), \tau_3(4,15,15)\}$ where the values inside of parenthesis represent execution time, period, and deadline respectively. T_s is a set of periodic tasks that can be scheduled with the EDF while unable to be scheduled with a fixed-priority scheduling algorithm. The EDF-CTI table for the given periodic tasks set is shown in Figure 2. An example of the EDF-CTI scheduling using an EDF-CTI table for the given T_s is shown in Figure 3.

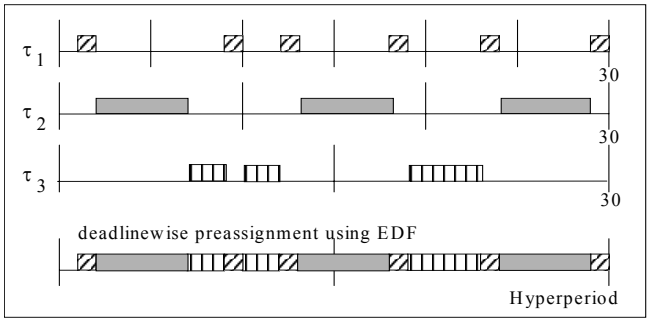

Figure 2. EDF-CTI Table for T_s.

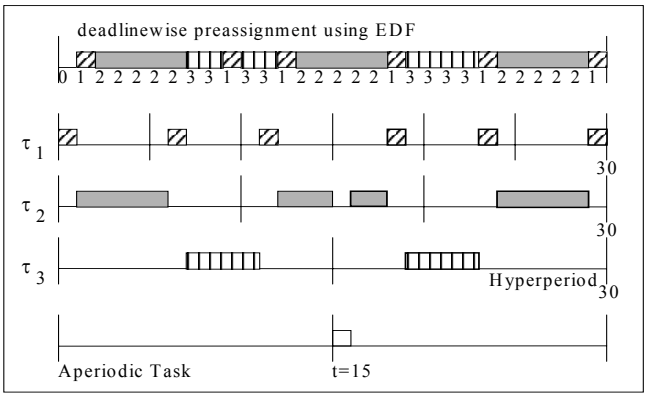

Figure 3. Aperiodic task service at t=15 using EDF-CTI algorithm.

In this example, when an aperiodic task A1 arrives at t=15, by the slack discriminant, the EDF-CTI algorithm indicates that the τ_2 is not a critical task at t=15. Consequently, A1 is assigned at that point immediately. Note that the slack discriminant ***f(t)*** for the dynamic-CTI algorithm indicates a slack if *CTI[t]* = 0 or *CP[CTI[t]] - CR[CTI[t]]* > 0, otherwise a critical task, where *t* denotes current scheduling time, *CTI* denotes CTI table (an array of integers, each element of which represents a slack or a periodic task), *CP[i]* and *CR[i]* (*i* represents an identifier of a periodic task) respectively denote all the computation processing done and all the computation requirement for each periodic task until *t*.

3. Simulation Result and Discussion

In this section, we observe the average response time for aperiodic requests. The performance of the dynamic-CTI algorithm will be compared to that of the fixed-CTI scheduling; all the combinations of a fixed- or dynamic-CTI table built off-line and an on-line fixed- or dynamic-priority assignment. In other words, we can combine an on-line algorithm for the CTI policy using fixed- or dynamic-priority with a fixed- or dynamic-CTI table, which has been built based on a fixed-priority or dynamic-priority scheduling. Thus, four different types of scheduling are available, i.e., a fixed-priority scheduling with a fixed-CTI table, a fixed-priority scheduling with a dynamic-CTI table, a dynamic-priority scheduling with a fixed-CTI table, and dynamic-priority scheduling with a dynamic-CTI table.

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.

In order to provide a fairly subjective observation ground for the simulation, we have constructed a model of simulation in a similar manner to that shown in the slack stealing algorithm. We arranged the periodic task set with 90% of CPU utilization to simulate high workload scheduling. This is summarized in Table 1.

	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 used in the simulations.

We also consider two cases of average execution time for aperiodic tasks, 3.5 and 20. In the following, the result of aperiodic response time for the above two cases is briefly discussed.

In Figure 4, as the workload of aperiodic tasks is increased, the average 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 5 shows that the average execution time for the given aperiodic tasks is 20. In this Figure, we observe the average response time for soft aperiodic tasks is dependent more on their average execution time; the larger average execution time for aperiodic tasks, the slower average response time for that tasks. The average response time for aperiodic requests is also affected by the on-line scheduling of periodic tasks. The simulation study, as a conclusion, shows that the average response time for aperiodic tasks is the smallest in case of EDF-EDF combination; the EDF on-line algorithm with a EDF-CTI table built off-line. Also, the response time of the aperiodic requests is faster than any other valid aperiodic task scheduling algorithms in fixed-priority systems, especially under the transient overload.

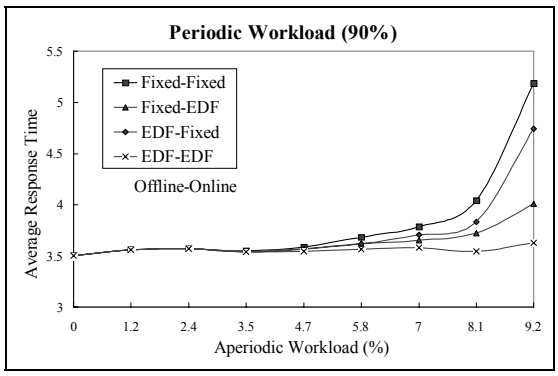

Figure 4. Average response time for aperiodic tasks when average execution time is 3.5.

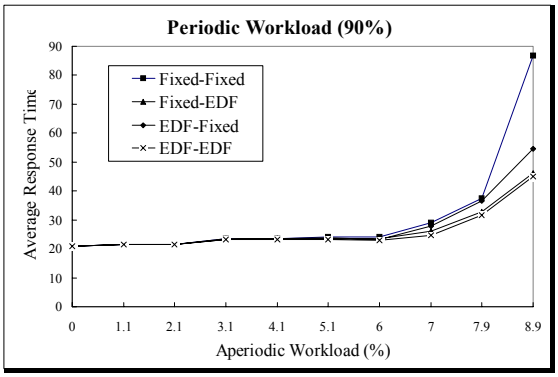

Figure 5. Average response time for aperiodic tasks when average execution time is 20.

Meanwhile, both of the fixed- and dynamic-CTI algorithms have some limitations such as the discrete unit time scheduling and the critical task misindicating problem. Misindicating of a non-critical task as a critical one may lead to the result that the algorithm can not find more available slacks at the scheduling point. Let us consider two periodic task T_1 and T_2, with T_1 having the higher priority. Suppose there is no aperiodic request initially. The first instant of T_1 finishes executing. An aperiodic task comes. Now, according to the fixed-CTI table,

T_2 may now be critical because the deadlinewisely preassigned scheduling is constructed with T_2 first, followed by T_1. But T_1 has already finished executing, and the fixed-CTI algorithm may not detect this. So instead of pushing T_2 back and scheduling the aperiodic request, the scheduler schedules T_2 instead, thinking that it is critical when in fact it is not critical because T_1 has finished and there may be more time later to execute T_2.

Due to the misindicating problem, our approach is not optimal, which means our algorithm can not find more available slacks. However, the computational complexity of our approach at off-line is O(NlogN) while that of Tia and Liu is $O(N^2)$ [13] where N is the number of periodic tasks in a hyperperiod. Moreover, our algorithm's time complexity to calculate the slack at on-line is O(1), which is the expense only for the discriminant of slacks, while that of the [13] is O(n) where n is the number of periodic tasks.

4. Extension of the Algorithm

In order to improve scheduling performance, our algorithm should eliminate the problems such as the misindicating of a critical task and the discrete unit time scheduling even though these are never seriously affects the overall scheduling. To develop an optimal scheduling in the CTI algorithm framework, we are under extending the dynamic-CTI scheduling algorithm.

The idea of this algorithm is that if the maximum slack provided by the CTI table is added to the maximum slack calculated at run-time, the algorithm should be an optimal. We will show the proofs of an optimality and how to overcome the limitations in the dynamic-CTI algorithm in another paper soon. We, however, briefly introduce the basic concept of this idea in this subsection.

We can calculate the maximum slack at the start point of time zone Z using the following formula

$$S_k(Z_{k-1}) = \underset{Z_{k-1}<t<Z_k}{S_k} + \sum_i \min\{(Cp_i - Cr_i), Cr_i\} \text{ --- } \boldsymbol{f1}$$

where k is an index for the time zone Z, i is an index of preassigned periodic tasks in the CTI table $[Z_{k-1}, Z_k]$. Note that Z is the time zone from the arrival or deadline of any periodic tasks' instance to the very next arrival or deadline of any periodic task instance. We can calculate the maximum slack in time zone Z while the periodic tasks are met their deadline. C_r is the computation requirement and C_p is the computation processing completed from the arrival time of ith periodic task instance to Z_{k-1}. In the right-hand side part of equality in formula ***f1***, S_k is a maximum slack provided by the CTI table off-line. The remaining part of the right-hand side calculated at run-time is the value of the periodic tasks' instances already being executed which will be a slack in the immediate following next zone Z. Figure 6 depicts how to partition the time zone Z based on the deadlines of the given periodic task set.

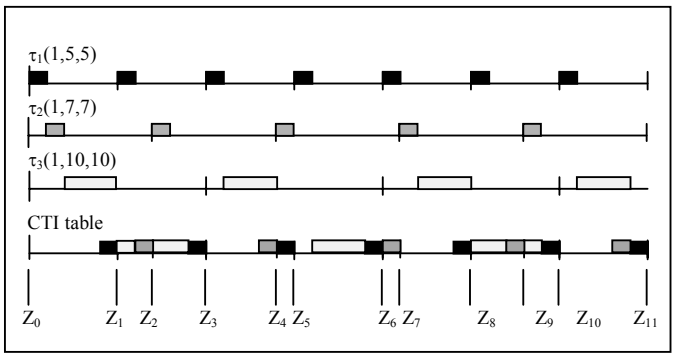

Figure 6. Example of partitioning time zone Z

The pseudocode of the extended EDF-CTI algorithm is shown in Figure 7. Note that A, P, and I are stand for an aperiodic task, a periodic task, and an idle time respectively.

5. Summary

```
when aperiodic task queue is not empty
   while (Sk(t) > 0) and (aperiodic task queue is not empty) do
     service aperiodic task(s) in Ak(t);
   Sk(t) := Sk(t) - Ak(t);

when periodic task arrive / complete
calculate Sk(t);
if periodic task in CTI is not served (i.e. Crk(t) > Cpk(t)) then
   while Crk(t) - Cpk(t) do
     service periodic task(s) by CTI table in Crk(t) - Cpk(t);
   Sk(t) := Sk(t);
if periodic task queue is not empty then
   service highest priority periodic task(s) in queue;
   Sk(t) := Sk(t) - Pk(t);  /* If Pk is  not used, it is added to the slack in the very next zone */
if no aperiodic task and periodic task then
   it's the idle time in Ik(t);
   Sk(t) := Sk(t) - Ik(t);
```

Figure 7. Pseudocode of the extended dynamic-CTI algorithm.

This paper discusses the problem of jointly scheduling hard deadline periodic and soft deadline aperiodic tasks in dynamic-priority systems. The deadlinewise preassignment based on dynamic-priority policy for a periodic task set in a single hyperperiod produces a scheduling table, called dynamic-CTI table, which is frequently referenced by the scheduler whether there are slacks for soft aperiodic tasks. It consequently keeps the benefits of the fixed-CTI algorithm that is simple to implement and offers remarkable scheduling predictability. The dynamic-CTI algorithm shows a good performance as much as the other fixed-CTI approaches in most cases and even better than the other soft aperiodic task schedulings in a transient overload. The proposed algorithm also relaxes the limitations of Chetto and Chetto's algorithm.

Even though our proposed algorithm is not optimal, it has considerable benefits such as the scheduling predictability and computational simplicity. We agree that the scheduling problem is, in some sense, a matter of tradeoff between simplicity and performance. In this respect, our approach more emphasizes upon the computational simplicity by introducing the CTI table.

References

[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] H.I.Kim, S.Y. Lee, J.W. Lee, and J.S. Kim, " An Aperiodic Task Scheduling Algorithm by Hybrid Priority Assignment in Real-Time Environments", Journal of the Korea Information Science Society, Vol. 22, No. 5, pp. 748-758, May, 1995

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