IEICE TRANS. INF. & SYST., VOL.E92–D, NO.7 JULY 2009

PAPER

Effective Scheduling Algorithms for I/O Blocking with a Multi-Frame Task Model*

Shan DING[†a)], ***Nonmember***, **Hiroyuki TOMIYAMA**[††b)], ***and*** **Hiroaki TAKADA**[††], ***Members***

SUMMARY A task that suspends itself to wait for an I/O completion or to wait for an event from another node in distributed environments is called an I/O blocking task. Conventional hard real-time scheduling theories use framework of rate monotonic analysis (RMA) to schedule such I/O blocking tasks. However, most of them are pessimistic. In this paper, we propose effective algorithms that can schedule a task set which has I/O blocking tasks under dynamic priority assignment. We present a new critical instant theorem for the multi-frame task set under dynamic priority assignment. The schedulability is analyzed under the new critical instant theorem. For the schedulability analysis, this paper presents saturation summation which is used to calculate the maximum interference function (MIF). With saturation summation, the schedulability of a task set having I/O blocking tasks can be analyzed more accurately. We propose an algorithm which is called Frame Laxity Monotonic Scheduling (FLMS). A genetic algorithm (GA) is also applied. From our experiments, we can conclude that FLMS can significantly reduce the calculation time, and GA can improve task schedulability ratio more than is possible with FLMS.

key words: *I/O blocking, multi-frame task model, schedulability analysis, laxity, genetic algorithm*

1. Introduction

A task that suspends itself to wait for an I/O completion or to wait for an event from another node in distributed environments is called an I/O blocking task. With the conventional framework of the rate monotonic analysis (RMA), an I/O blocking task cannot be scheduled effectively. The reason for this is that the critical instant theorem does not follow its original form when a task has I/O blocking, and various analysis techniques derived from the theorem cannot be accordingly applied [1].

Takada [1] has proposed an approach using multi-frame task model to schedule a set of tasks including I/O blocking [2]. According to this approach, a task having I/O blocking is divided into two frames f_1 and f_2, whose maximum execution times are C_1 and C_2, respectively, one frame applicable before the blocking and the other after the blocking. The whole task period is denoted as T, and the periods of f_1 and f_2 are denoted as T_1 and T_2, respectevely, where $T = T_1 + T_2$. f_1 is started periodically within the period of T_1 and an I/O operation is requested after f_1 is completed. Then, f_2 is started immediately after the I/O operation is completed. Here, we assume that periods T_1 and T_2 are equal to deadlines of f_1 and f_2, respectively.

In [1], a method is considered by which the deadline of a frame can be allocated so that the maximum schedulability of lower priority frames can be achieved. If subtraction of the execution time from the deadline can be described as the slack of a frame, a simple allocation method of deadlines is to make the slacks of the two frames equal. In other words, $T_1 - C_1 = T_2 - C_2$. In addition to the $T = T_1 + T_2$ formula, T_1 can be calculated by the formula $T_1 = (T + C_1 - C_2)/2$. This method is described as an equal slack allocation method. Although this method is not the optimal method, compared to conventional approaches, it does provide a useful way of improving the schedulability of I/O blocking tasks.

Endo et al. have discussed the critical instant theorem for generalized multi-frame tasks (GMF) with the dynamic priority assignment [3]. By use of the maximum interference function (MIF) [4], they have calculated the interference time caused by high priority tasks within the duration time interval t. Based on this theorem, they discussed the necessary and sufficient condition for the schedulability of a set of GMF tasks under dynamic priority assignment. As described later, our algorithms presented in this paper are based on their work.

Iiyama et al. [5] have applied the GMF model to engine management systems. According to their work, the degree of schedulablity of a frame is captured by summation of the maximum total interference times requested by all higher priority frames within the duration of time t. This summation can be calculated by "adding up" the MIFs of all higher priority frames. However, the time which obtained by normal arithmetic addition is longer than actual interference time, so they have proposed saturation summation for greater precision. We also employ saturation summation to improve the calculation of MIF in our algorithms.

As above described, scheduling of a set of tasks which has I/O blocking, allocation of the deadline and assignment of priorities to the frames simultaneously, remain as an open issue. Specifically, no approach exists to assign arbitrary priorities within a general multi-frame model. We propose an algorithm which is called Frame Laxity Monotonic Scheduling (FLMS). A genetic algorithm is also applied. As a result of our experiments, we have concluded that the genetic algorithm we proposed significantly improve task schedulability ratio more than is possible with FLMS.

The remainder of this paper is organized as follows: In

Manuscript received October 3, 2008.
Manuscript revised February 16, 2009.

[†]The author is with the College of Information Science and Engineering, Northeastern University, Shenyang, P. R. China.
[††]The authors are with the Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8603 Japan.

*This paper was presented at RTCSA 2007.

a) E-mail: dingshan@ise.neu.edu.cn
b) E-mail: tomiyama, hiro@ertl.jp

DOI: 10.1587/transinf.E92.D.1412

Sect. 2 the multi-frame task model is defined, and an illustrative example is given. Section 3 discusses the necessary and sufficient condition for the schedulability of a set GMF tasks with dynamic priority assignment. Section 4 discusses in detail our approaches. In Sect. 5, experimental tests are carried out, and some concluding remarks follow in Sect. 6.

2. The Multi-Frame Task and an Illustrative Example

In this section, we give a brief description of the multi-frame task model under dynamic priority assignments, then give a simple example to show difference priorities can improve schedulability of a task set which has I/O blocking tasks.

2.1 The Multi-Frame Task Model

The multi-frame task model is effective for tasks whose execution times are periodically changed according to a specified pattern. Under this model, each period with the Liu and Laylaud (L&L) task model is called a frame, and the task maximum execution time in each frame is changed with a larger cycle. In other words, a set of tasks is divided into some frames with the same period, and the maximum execution time of a task is set for the frame which it belongs to. A task is set at the beginning of the frame, and the deadline is the end of the frame. The generalized multi-frame task model is an extension of the multi-frame task model in the following two directions: (1) different frames may have different periods and (2) the deadline for a frame may be different from the end of the frame [6]. From now on, we will describe the generalized multi-frame task model simply as a multi-frame task model.

In addition to priorities for frames under multi-frame model, a multi-frame task T_i consisting of N_i frames is characterized by a sequence of 4-tuples $((C_i^0, D_i^0, P_i^0, L_i^0), \ldots, (C_i^{N_i-1}, D_i^{N_i-1}, P_i^{N_i-1}, L_i^{N_i-1}))$, where C_i^j, D_i^j, P_i^j and L_i^j represent the maximum execution time of the jth frame of T_i, its relative deadline, its priority and its length, which is the minimum separation time between the arrival time of the jth frame and the following frame. In order to maintain the frame separation properties, we assume that $D_i^j \leq L_i^j$, holds for all j $(0 \leq j \leq N_i - 1)$ [4].

An I/O blocking task is shown in Fig. 1. The I/O blocking task is divided into frame1 which is before the blocking and frame2 which is after the blocking. Frame1 and frame2 are expressed as (C1, D1, P1, L1) and (C2, D2, P2, L2), respectively. D is the total deadline of the I/O blocking task. In this paper, we assume that D is equal to the period of the I/O blocking task. In consideration of the I/O blocking task constraints, three equations are applied as follows.

$$L1 = D1 + B$$
$$D = L1 + D2$$
$$P = L1 + L2$$

The focus of our research is to find an optimal scheduling method by adjustment of priorities and deadlines of frames. To a set of period tasks according to the L&L task model, the Deadline Monotonic (DM) [7] priority assignment is optimal, meaning that if any static priority scheduling algorithm can schedule a set of tasks with deadlines that are unequal to their periods, then DM will also schedule the task set. However, DM is not the optimal scheduling method within multi-frame task model. To demonstrate this, we show an example below.

2.2 An Illustrative Example

A multi-frame task τ_m consists of two frames ((3, 3, P1, 3), (2, 5, P2, 5)). There is another period task τ, the execution time, the deadline, the relative priority and the period of task τ are 3, 6, P3 and 8, respectively. According to DM, the two frames of task τ_m is allocated higher priority and task τ with lower priority. Used with this kind of priority assignment, the task set is not schedulable, as shown in Fig. 2 (a).

However, if we assign $\tau_{m,0}$ the highest priority, τ middle priority, $\tau_{m,1}$ the lowest priority respectively, the task set is schedulable, as shown in Fig. 2 (b).

This illustrates that DM can not effectively schedule a GMF task set. Since the release time of frames is generated by dynamic priority assignment, the problem becomes NP-hard [8]. On the other hand, an I/O blocking task allows preemption tends to make the problems easier. We consider that dynamic priority assignment for frames can improve the schedulability of a set of tasks.

2.3 Related Work

The I/O performance has been considered in some applications such as distributed systems, multiprocessor or

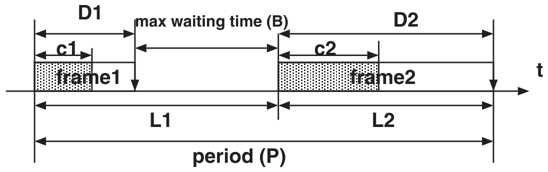

Fig. 1 An I/O blocking task.

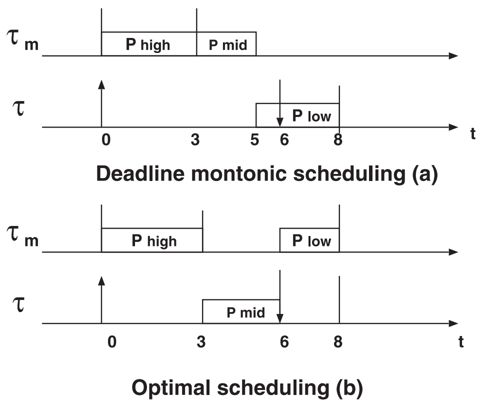

Fig. 2 An example of DM.

multiple-disk systems [8]–[11] by other approaches. A.L. Narasimha Reddy and Prithviraj Banerjee have been improved the I/O performance by building an I/O subsystem with multiple disks [9]. Specifically, they considered disk synchronization, data declustering/disk striping, and a combination of both these approaches. The effects of block size and other parameters of the system have been evaluated. In [10], several design options in designing an I/O system have been studied. Trace driven simulations have been used to study a disk system with nonvolatile cache. It was shown that decoupling the cache block size and fetch size yielded significant performance benefits. Also, A.L. Narasimha Reddy [11] studied the I/O behavior of some scientific applications, a subset of perfect bechmarks, executing on a multiprocessor. The aim of this study is to explore the various patterns of I/O access of large scientific applications and to understand the impact of this observed behavior on the I/O subsystem architecture.

In distributed systems, the lack of global information about data transfer between clients and servers make implementation of parallel I/O a challenging task. J. Wu et al. have proposed two distributed algorithms for scheduling data transfer in parallel I/O with non-uniform data sizes [12]. Their simulations show that two algorithms are suitable for parallel I/O with data transfer traffic.

3. Schedulability Analysis

3.1 Critical Instant Theorem

With L&L task model, a critical instant of any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks [13]. In case of a GMF task set which has non-AM (Accumulative Monotonic) tasks, a critical instant for the frame cannot be determined independently of the execution time of the higher priority frames. For example, consider an I/O blocking task $\tau_m(f_1, f_2)$ = ((3, 3, P1, 3), (1, 5, P2, 5)) and a task τ = (2, 2, P3, 5). Assuming P1 > P3 > P2, if we consider a critical instant of the task set with multi-frame model as above, a critical instant of the lowest priority frame f_2 occurs when f_2 is requested simultaneously with requests for τ. As shown in Fig. 3 (a), the response time of f_2 is 3.

However, the worst case response time of f_2 is as shown in Fig. 3 (b), and a critical instant of f_2 defined as above is not the worst case response time of f_2. Even if f_1 is executed at time 0, it can be postponed by the length of f_2. Based on the preemptive static priority scheduling theorem, we give a brief description of the new critical instant theorem for a GMF model [4].

Definition 1 (Critical Instant Candidates): The critical instant candidates of a GMF task are defined as the instants at which a frame of the task is requested simultaneously with any one of the frames with higher priority. Their succeeding frames are requested with their minimum separations, and each frame is executed for a maximum execution time.

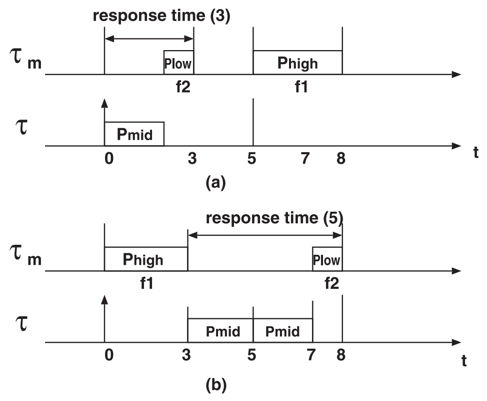

Fig. 3 An example of the critical instant theorem.

On the basis of such a definition, we can describe the critical instant theorem for a GMF task as follows.

Theorem 1 (Critical Instant Theorem): A critical instant of a frame of a GMF task occurs at one of the critical instant candidates for the task. In other words, one of the critical instant candidates is a critical instant of the frame.

Proof. All frames which are in a frame set U have their priorities. We assume that when frame F with priority i of the task T is released at time t and completes at time t_{end}, the worst case response time of frame F is caused. In this situation, a frame is sure to be executed with higher priority than the priority of F between time t and t_{end}. The time at which there is no frame with higher priority has been executed before time t is assumed to be t_0. Because any frame is not executed at time 0, t_0 is sure to exist. It is assumed that F is the first arrival of T after t_0. Even if we change the arrival time t of frame F to t_0, it still completes at time t since there is at least one frame with priority that is higher than i is executed.

Assuming that for each frame with priority that is higher than i, we move its first arrival time after t_0 to t_0. In this case, the completion time of frame F might be delayed to $t_{end'}$. Moreover, their succeeding frames are requested early with their minimum separations, and each frame is executed for a maximum execution time. The completion time of frame F is delayed further to time $t_{end''}$. Since time t is one of critical instant candidates of T, the response time of frame F is longer than the first situation.

When we assume that if the worst case response time of frame F is caused in the first situation, the worst case response time of frame F is caused too when the critical instant candidates for frame F are requested.

Next, assuming that frame F is not the first arrival of T after t_0, we prove it as follows. If frame F with priority i is the kth frame of the arrival of T after t_0, we assume that the worst case response time of frame F is caused, and it completes at time t_{end}. As above proof, the time which a frame with higher priority has not been executed before t is assumed to be t_0. It is sure that t_0 exists. We assume that frame F' with higher priority j is the first arrival of T

after t_0 ($j \geq i$). The arrival time of F' is t'. It completes at time $t_{end'}$. Even if we change at the arrival time of F' from t' to t_0, it still completes at time $t_{end'}$ since there is at least one frame with priority that is higher for i is executed. Therefore, frame F still completes at time t_{end}.

Assuming that for each frame with priority that is higher than i, we move its first arrival time after t_0 to t_0. In this case, the completion time of frame F might be delayed to $t'_{end'}$. Moreover, their succeeding frames are requested early with their minimum separations, and each frame is executed for a maximum execution time. The completion time of frame F is delayed further to time $t'_{end''}$. Since time t is one of critical instant candidates of T, the response time of frame F is longer than the first situation.

When we assume that frame F with priority i is the kth frame of the arrival of T_i after t_0, the worst case response time of F is caused. If there exists a frame between F and F' with priority that is lower than i, t_0 exists between t and t'. t is one of critical instant candidates of T.

Based on the new critical instant theorem, a frame in the task set is schedulable when critical instant of the frame is smaller than its deadline.

3.2 Maximum Interference Function (MIF)

In order to decide the feasibility of a set of multi-frame tasks effectively, a maximum interference function (MIF) is proposed in [4]. The MIF of a task is a function presenting the maximum time during which the execution of the task interferes with the execution of the lower priority tasks within the duration time interval t.

The MIF $M_i(t)$ of a multi-frame task T_i is represented as follows,where I_i^k is called the interference function (IF) and represents the maximum time during which T_i interferes with lower priority tasks during t after the arrival of the kth frame of T_i.

$$M_i(t) = \max_{0 \leq k \leq N_i - 1} I_i^k(t)$$

It can be transformed into the following form.

$$I_i^k(t) = \sum_{h=k}^{k+J-1} C_i^{h mod N_i} + min\left(C_i^{(k+J) mod N_i}, t - \sum_{h=k}^{k+J-1} L_i^{h mod N_i} \right)$$

Where J is the maximum integer that satisfies the following formula.

$$\sum_{h=k}^{k+J-1} L_i^{h mod N_i} \leq t$$

The MIF of a multi-frame task is the maximum value of each interference function (IF) of frames when interference frames are released simultaneously. Consider a task consisting of three frames: τ_i = ((1, 4, P1, 4), (2, 4, P2, 4), (3, 4, P3, 4)), Fig. 4 illustrates the IFs of three kinds of conditions and the MIF of τ_i. The MIF of the multi-frame task τ_i is the uppermost connected lines of the three IFs.

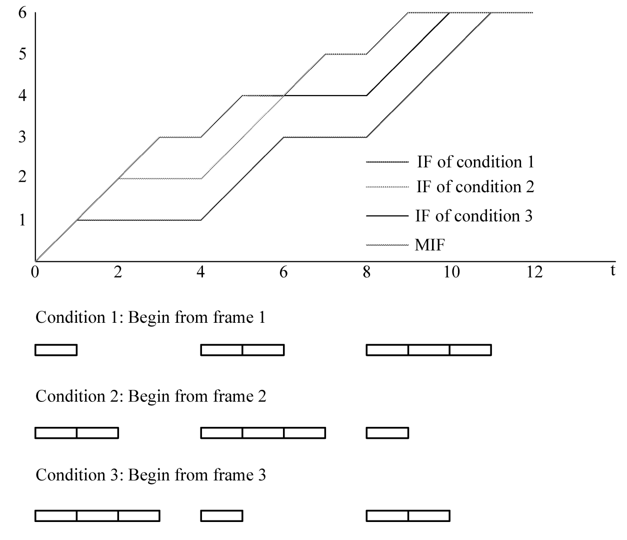

Fig. 4 The MIF of a GMF task.

From the definition of the MIF, the necessary and sufficient condition for the schedulability of a multi-frame task T_i can be given as follows.

A multi-frame task T_i is schedulable under a GMF model if and only if;

$$\exists t, 0 \leq t \leq D_i^k, \sum_{j=1}^{i-1} M_j(t) + C_i^t \leq t \tag{1}$$

According to the above formula, we can obtain schedulabe conditions for a task during time interval t. Since the interference time caused by one higher priority multi-frame task can be calculated, the total interference time caused by one or more than higher priority multi-frame tasks can be intuitively calculated by arithmetically adding of them. In fact, for a set of multi-frame tasks that has one or more tasks, the result of the arithmetical addition of MIFs is longer than the actual interference time caused by all the higher priority tasks. Saturation summation has been applied to improve MIF for a task set.

3.3 Saturation Summation

For a set of tasks, the degree of schedulable of a frame is captured by total interference time requested by all higher priority frames within the duration of time interval t. As described above, the time is calculated by arithmetical addition of MIFs is longer than the actual interference time of all higher priority tasks.

Assume that two multi-frame tasks exist, τ_1 = ((1, 8, 4, 8), (2, 8, 3, 8)) and τ_2 = ((3, 8, 2, 8), (2, 8, 1, 8)). $M_1(t)$ and $M_2(t)$ are MIFs of τ_1 and τ_2 respectively. The MIF of τ_1 and τ_2 is shown in Fig. 5 (a) (i.e., $M(t) = M_1(t) + M_2(t)$). $M_s(t)$ constitutes the actual interference time affected by the two multi-frame task τ_1 and τ_2. From Fig. 5, the value of $M(t)$ is greater than that of $M_s(t)$ during the two intervals of time (0, 4) and (8, 11), which means that $M(t)$ is longer than the actual interference time in the same two intervals. In

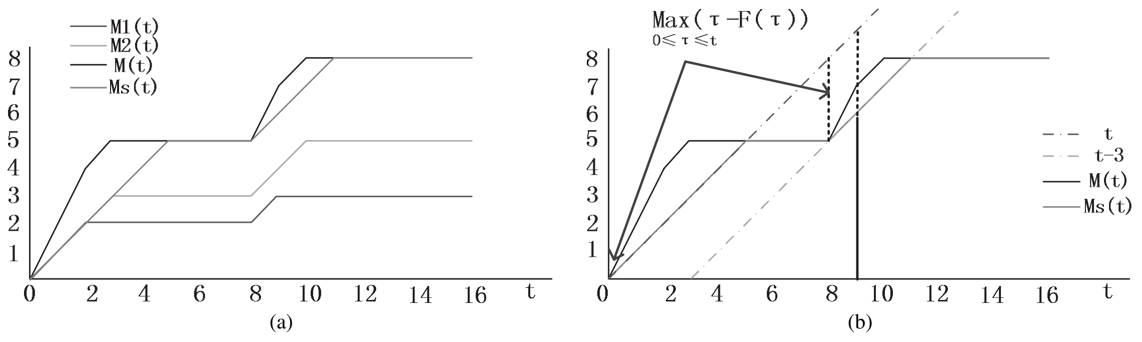

Fig. 5 The saturation summation.

fact, MIF is impossible to increase like $M(t)$ during the two intervals of time, because only one frame is executed during the intervals. For this reason, $M(t)$ must be converted into $M_s(t)$ by use of a saturation conversion.

Definition 2 (Saturation Conversion): Given a function $F(t)$ that depends on time with a nonnegative integer gradient, a new function $F_s(t)$ can be obtained on the basis of the following formulas. This is called a saturation conversion.

$$F_s(t) = t - \max_{0\le\tau\le t}(\tau - F(\tau))$$

As for the definition of saturation conversion, the function that is converted from the arithmetical addition of MIFs by a process of saturation conversion can be described as saturation summation, which can be expressed as follows:

$$\bigoplus_{j=1}^{i-1} M_j(t) = t - \max_{0\le\tau\le t}\left(\tau - \sum_{j=1}^{i-1} M_j(\tau)\right) \tag{2}$$

Figure 5 (b) illustrates saturation summation applied to the same set of tasks in Fig. 5 (a).

- In the interval of time [0, 5], because $\sum_1^3 M_j(\tau) \ge \tau$,

$$\max_{0\le\tau\le 4}\left(\tau - \sum_{j=1}^{i-1} M_j(\tau)\right) = 0, \implies \bigoplus_{j=1}^{i-1} M_j(t) = t;$$
$$(0 \le t \le 4)$$

- In the interval of time [8, 11], because,

$$\max_{8\le\tau\le 11}\left(\tau - \sum_{j=1}^{i-1} M_j(\tau)\right) = 3, \implies \bigoplus_{j=1}^{i-1} M_j(t) = t - 3;$$
$$(8 \le t \le 11)$$

Thus, saturation summation of MIF can compensate for a discrepancy and calculate interference time precisely.

The necessary and sufficient condition for the schedulability of a kth frame of a multi-frame task T_i must be reformulated for saturation summation of MIF.

According to the definition of saturation summation, the inequality 1 can be transformed into the following inequality.

$$\exists t, 0 \le t \le D_i^k, \ \bigoplus_{j=1}^{i-1} M_j(t) + C_i^t \le t \tag{3}$$

Since saturation summation is a monotonic non-decreasing function, the laxity of a multi-frame task T_i is at the maximum at a time t is equal to the task deadline. Based on the above discussions, it is easy to observe that saturation summation of MIF can only be inspected at the time of the deadline t, but not at each $t' < t$. The necessary and sufficient condition for the schedulability of the kth frame of a multi-frame task T_i is described below:

Theorem 2: The kth frame of a multi-frame task T_i can be schedulable if and only if

$$\bigoplus_{j=1}^{i-1} M_j(D_i^k) + C_i^t \le D_i^k$$

This theorem is proved by demonstrating the two inequalities (1) and (3). When inequailty (1) is satisfied, then inequailty (3) is also satisfied. The two inequalities are equivalent to the schedulabe conditions of the kth frame of a multi-frame task T_i. The proof of this is described in details elsewhere [5].

4. Scheduling Algorithms

When priorities for a frame are assigned sequentially, the worst-case complexity of searching the frame deadline with n frames is of a higher order than O(n!). It took several days to calculate a 10 frames problem (4 I/O blocking tasks and 2 non-I/O blocking tasks). The algorithm is practically unusable for most real-time applications.

4.1 Frame Laxity Monotonic Scheduling (FLMS)

In this section, we present a minimum laxity first dynamic scheduling algorithm based on a greedy approach such as [14]. Laxity is a measure of the flexibility available for scheduling a task. The basic idea is that the shorter laxity of frame is, the higher priority assigned to the frame. In general, laxity X_i of a non-I/O blocking ith frame is the maximum time that the frame can be delayed on its activation to complete within its deadline. For an I/O blocking task

```
Algorithm: FindUrgencyFrame
Input:  The minimum laxity frame $f_{min}$;
Output:  An urgency frame $f_u$ if it exists;
1. For each ith unselected frame in the set of frames F;{
2.     Assume the ith frame is in the jth task;
3.     if (MIF$f_{min}$ + The execution time of the ith frame ≥ The deadline of the jth task) {
4.         The urgency frame $f_u$ is the ith frame;
5.         Return $f_u$;
6.     }
7.}
```

Fig. 6 Find an "urgency" frame.

τ_k(including two frames f and f'), we have defined laxity X_f of the frame f as follows:

$$X_f = \begin{cases} D_k - WCIT(f) - L_{f'} - C_f - B(f) \\ \qquad\qquad \text{if } P_{f'} \text{ is always fixed.} \\ (D_k - WCIT(f) - C_f - C_{f'} - B_k)/2 \\ \qquad\qquad \text{if } P_{f'} \, and P_f \text{ are not fixed.} \end{cases}$$

Where, D_k is a task deadline, $WCIT(f)$ is amount of time during which interference is caused by the execution of higher priority frames in worst case, as calculated by means of theorem 1. $L_{f'}$ is the length of frame f'. $B(f)$ is a function, when frame f is the frame applicable before I/O blocking, $B(f)$ returns a value of a maximum waiting time (B_k) of the task; otherwise, $B(f)$ returns zero.

According to the above formula, if priorities of f' and f are not assigned, frame laxity values of f' and f are equal to the average equivalent of the laxity of the I/O blocking task. Note that, B_k is the maximum waiting time.

If more than one frame have the same laxity, the algorithm randomly selects the one to which higher priority should be given.

To consider the I/O blocking task mechanism, we improved FLMS based on the following strategies.

- FLMS takes into consideration the execution time of a frame. However it can not ensure that other frames do not miss deadlines. Consider two frames τ_1 (1, 100, P_1, 100) and τ_2 (52, 150, P_2, 150). The interference time caused by higher priority frames is 50. The laxity of τ_1 and τ_2 are 49 and 48, respectively. According to FLMS, τ_2 is assigned a higher priority. But τ_1 is unschedulable. This example illustrates that a frame with minimum laxity is not necessary an "urgency" frame that must be executed in the first instant, the "urgency" frame does not always exist. We compare the total of interference time and execution time of the frame with minimum laxity with the deadlines of other frames so as to find an "urgency" frame. If the deadline of a frame is smaller than the total value, the frame is deemed to be an "urgency" frame, and must be assigned higher priority. Otherwise, the frame with the minimum laxity is assigned the higher priority (line 12). Figure 6 illustrates how to find an "urgency" frame, where $MIF_{f_{min}}$ is the total of interference time.

```
Algorithm: Frame Laxity Monotonic Scheduling
Input:  A set of frames F. p is the highest priority;
Output:  Priority of frames, deadline of frames;
1. while (F!= NULL) loop
2.     For each frame $f_i$ in the set F;{
3.         Assume that the frame $f_i$ is in task $t_j$;
4.         Generate the IFs of the higher priority tasks that does not have IF of the task $t_j$;
5.         Calculate the laxity of $f_i$;
6.         if (The laxity of $f_i$ is minimum) then $f_{min}$ = $f_i$;
7.     }
8.     if (The frame is unschedulable) then {
9.         Return fail;
10.    }else{
11.        if (FindUrgencyframe($f_{min}$)=$f_u$) $f_{min}$ = $f_u$;
12.        Set priority of frame $f_{min}$ to p; F = F − $f_{min}$;
13.        Set deadline of frame $f_{min}$ to MIF($f_{min}$) + execution_time($f_{min}$);
14.        Decrease task priority p;
15.    }
16.    if ((Frame $F_{min}$, $f_2$ in the I/O blocking task $t_i$)&($f_2 \notin$F)){
17.        The equal slack allocation (the I/O blocking task $t_i$);
18.    }
19. end loop;
```

Fig. 7 Frame laxity monotonic scheduling. (FLMS)

- The two frames that are in an I/O blocking task do not mutually interfere with each other regardless of the priority of the two frames. In other words, when calculating interference time of the *i*th frame, it is unnecessary to generate the IF of the multi-frame task to which the *i*th frame belongs (line 4).
- Whenever a frame fails to meet its deadline, the equal slack allocation proposed in [3] is used to maximize the schedulability of lower priority frames. If the priority of two frames of an I/O blocking task is greater than that of a frame that fails to meet its deadline, the deadline of the two frames is adjusted by this method (line 17).

According to FLMS mechanism, the minimum laxity frame in a set of frames in which frames are not assigned priorities is taken into account (from line 2 to line 7). If the frame is unschedulable, showing the whole frame set is unschedulable, the algorithm is stop at this step (line 9). Otherwise, finding an urgency frame is performed at line 11. If the urgency frame is existed, it is used to replace the minimum laxity frame. From line 11 to line 14, priority of the frame we selected is assigned and deadline of the frame is determined. At the end, the equal slack allocation is performed if the frame we selected is in an I/O blocking task (line 17). Note that priority of a frame is not equal to that of another frame, since the relevant MIF calculation becomes more complex. The method is summarized in Fig. 7.

4.2 Genetic Algorithm for I/O Blocking

Our GA requires the definition of a set of genetic operations and an evaluation function as follows:

4.2.1 Coding Individual

Each frame is denoted with frame identifier (FrameID). Based on the frame identifier, we can find whether an I/O task that the frame belongs to. Priorities of frames can be encoded by a string where the *i*th number in string is priority of the *i*th frame. Another string is used for encoding deadlines of frames. The lengths of two strings are the number of frames. The deadline of a frame with the highest priority is equal to the maximum execution time of the frame. Priorities and deadlines are generated randomly in their possible ranges. According to the FrameID, priority and deadline of the frame are easy to find.

4.2.2 Generate Initial Generation

Individuals are generated randomly as many as population size. The fitness value of an individual is calculated by an evaluation function. When the same individual is generated, it is deleted and a new different individual is obtained by using mutation operation. Then individuals in the generation will be sorted in ascending order.

4.2.3 Evaluation Strategy

The fitness value of an individual is the number of frames that are schedulable in the task set. The schedulability of a frame is calculated according to the new critical instant theorem.

4.2.4 Crossover

The crossover operation is held between two parents. Two selected individuals are called parent-A and parent-B, and assume the fitness value of parent-A is higher or equal to that of parent-B. Every crossover results in one offspring by randomly choosing one of two suboperations. One is to exchange priorities of some genes. This mechanism can be carried out by cyclic permutation operation. The same performance was described elsewhere [15] in details.

Another operation is to change deadlines of some genes by the following steps:

1. For a gene d_a in parent-A and the gene d_b at the same position as d_a in parent-B, if $d_a = d_b$ then set d_a to the same position in the offspring.

2. Let the number of unset genes in the offspring be m.

   (a) if m ≤ 4 then for every unset positions in the offspring, set the average value of genes at the same position in parents to the offspring.
   (b) if m > 4 then: select $[m + 1]/4$ genes from parent-A randomly and set them to the offspring. If the deadline of a frame in an I/O blocking task was determined, the deadline of another frame in the same I/O blocking task is also determined. Values of unset genes in offspring are copied from genes at the some position in parent-B.

At the end of crossover operation, let the deadline of a frame with the highest priority equal to the maximum execution time of the frame.

4.2.5 Mutation

The mutation operation is in an individual. We also conduct mutation operation by randomly choosing one of two suboperations. One is to choose an I/O blocking task randomly, change deadlines of two frames randomly. Another is to choose an I/O blocking task randomly, set the average value of deadlines in two frames.

4.2.6 Other Operations

Selection and differentiation of the same individual are performed as described elsewhere [16] in details.

Parameters for GA were set as follows. Population size was five times as much as the number of frames. The probability of mutation was 0.2. The number of generations was 1000.

5. Experiments

This section presents a set of experiments demonstrating the effectiveness of our approach. The two developed algorithms are implemented using C language executed on a 2.53 GHz Pentium4 processor with a 512 MB RAM. We have not conducted a comparison with other algorithms. Because I/O blocking with a multi-frame task model is different from L&L task model and two algorithms we propose are also based on the new critical instant theorem. We conducted tests on four task sets consisting of 5, 10, 15 and 20 frames, respectively. 20 samples were generated randomly for each task set. The main parameters of a task are listed in Table 1. The minimum execution time and the maximum execution time of frame are 10 and 100 time units, respectively.

The number of frames, I/O blocking tasks and non-I/O blocking tasks in the four task sets are listed in Table 1. The task deadlines and the blocking waiting times were generated randomly between the minimum time and the maximum time of the deadline and the waiting time, respectively.

The data reported in Figs. 8 and 9 are the average values of 20 samples obtained by FLMS and GA for each data set,

Table 1 Parameters of task.

Name of data sets	Set 1	Set 2	Set 3	Set 4
Number of frames	5	10	15	20
Number of I/O blocking tasks	2	4	7	9
Number of non-I/O blocking tasks	1	2	1	2
Minimum deadline for task	100	400	700	1300
Maximum deadline for task	500	1000	1300	2000
Minimum waiting time	20	20	20	80
Maximum waiting time	90	90	90	150

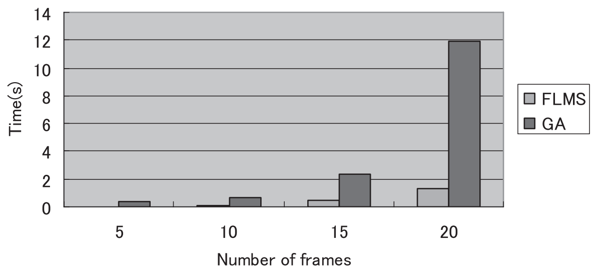

Fig. 8 The time required comparison between FLMS and GA.

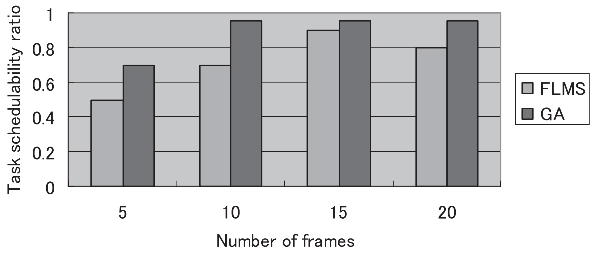

Fig. 9 Task schedulability ratio comparison between FLMS and GA.

respectively. Figure 8 presents the average time required for algorithm computation comparison between FLMS and GA. FLMS is better than that of GA for all 4 data sets. As above described, if priorities for frames are assigned sequentially, it took several days to calculate a 10 frames problem. The maximum computing time for the biggest problem by GA is 12 seconds. Compared with several days, it is not a serious problem for 12 seconds.

Another reason why computation time of FLMS is less than that of GA is, GA is a kind of evaluation algorithm. When the individual's chromosome changes by genetic operations, it needs to calculate interference function again. FLMS only needs calculate interference functions according to greedy approach. This also indicates that these strategies of FLMS are success and reduction of the number of interference functions leads up to significant reduction of calculation time of the saturation summation.

Figure 9 illustrates the task schedulability ratio comparison between FLMS and GA. The task schedulability ratio is the proportion of the schedulable data samples to each data set calculated by one kind of algorithms. Figure 9 demonstrates significant improvement of GA over FLMS. From experiment, we can conclude that FLMS can significantly reduce the calculation time, however the GA can improve task schedulability ratio more than is possible with FLMS. Experimental results show that our developed methods are able to effectively produce good quality results. Because the assumption of our range of experiment parameters is realistic, we can conclude that our approach is sufficiently effective for practical purposes.

6. Conclusions

In this paper, effective algorithms that can schedule a set of tasks which has I/O blocking under dynamic priority assignment are proposed. We have presented a new critical instant theorem necessary for a multi-frame task set with arbitrary priority assignment. An approach to schedulability analysis based on the critical instant theorem is also presented. In addition, we have proposed an FLMS algorithm. A genetic algorithm is also applied. From our experiments, we can conclude that the FLMS can significantly reduce the time of the calculation time, and GA can improve task schedulability ratio more than is possible with FLMS.

References

[1] H. Takada, "Schedulability a task including I/O blockings with the multiframe task model," 19th IEEE Real-Time Systems Symposium WIP, Madrid, Spain, Dec. 1998.
[2] A.K. Mok and D. Chen, "A multiframe model for real-time tasks," Proc. Real-Time Systems Symposium, pp.22–29, Dec. 1996.
[3] Y. Endo and H. Takada, "Studies on the schedulability analysis of tasks set with I/O blockings," IPSJ SLDM, pp.105–111, March 2002.
[4] H. Takada and K. Sakamura, "Schedulability of generalized multiframe tasks set under static priority assignment," Proc. Real-Time Computing Systems and applications, pp.80–86, Oct. 1997.
[5] S. Iiyama, H. Takada, and H. Suganuma, "A schedulability analyzing method for engine management systems," J. IPSJ Trans. Advanced Computing Systems, vol.43, no.6, pp.1715–1724, June 2002.
[6] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," Real-Time Systems, vol.17, no.1, pp.5–22, 1999.
[7] J.Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," Perform. Eval., vol.2, no.4, pp.237–250, Dec. 1982.
[8] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," Ann. Discrete Math., vol.5, pp.287–326, 1979.
[9] A.L. Narasimha Reddy and O. Banerjee, "An evaluation of multipledisk I/O systems," IEEE Trans. Comput., vol.38, no.12, pp.1680–1690, Dec. 1989.
[10] A.L. Narasimha Reddy, "A study of I/O system organizations," Proc. 19th Annual International Symposium on Computer Architecture, pp.19–21, May 1992.
[11] A.L. Narasimha Reddy and O. Banerjee, "A study of I/O behavior of perfect benchmarks on a multiprocessor," Proc. 17th Annual International Symposium on Computer Architecture, pp.28–31, Seattle, WA, USA, May 1990.
[12] J. Wu, Y. Lin, and P. Liu, "Efficient distributed algorithms for parallel I/O scheduling," Proc. 11th International Conference on Parallel and Distributed System, pp.20–22, Fukuoka, July 2005.
[13] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," J. ACM, vol.20, no.1. pp.46–61, 1973.
[14] D. Stewart and P. Khosla, "Real-time scheduling of sensorbased control systems," Proc. IFAC/IFIP Workshop, pp.139–144, Tarrytown, NY, May 1991.
[15] S. Ding, H. Tomiyama, and H. Takada, "An effective GA-based scheduling algorithm for FlexRay systems," IEICE Trans. Inf. & Syst., vol.E91-D, no.8, pp.2115–2123, Aug. 2008.
[16] S. Ding and N. Ishii, "Determining feature weights of pattern classification by using rough genetic algorithm with fuzzy similarity measure," J. Japan Society for Fuzzy Theory and Systems, vol.14, no.3, pp.310–319, 2002.

Shan Ding received the B.E. degree in Department of Computer Science and Application in Northeastern University, Shenyang, China, in 1989, and the M.E. degree in Department of Electrical and Computer Engineering from Nagoya Institute of Technology of Japan, in 1999. He received his Ph.D. degree in computer science from Nagoya Institute of Technology in 2003. From Sept. 2004 to Aug. 2006, he was a visiting postdoctoral researcher in the Department of Information Engineering, the Graduate School of Information Science, Nagoya University, where he was financially supported by JSPS Postdoctoral Fellowship for Foreign Researchers. He is currently an Associate Professor in College of Information Science and Engineering, Northeastern University, P. R. China. His current research interests include algorithm design and analysis, real-time scheduling theory, and embedded system design.

Hiroyuki Tomiyama received his Ph.D. degree in computer science from Kyushu University in 1999. From 1999 to 2001, he was a visiting postdoctoral researcher with the Center of Embedded Computer Systems, University of California, Irvine, where he was financially supported by JSPS Postdoctoral Fellowship for Research Abroad. From 2001 to 2003, he was a researcher at the Institute of Systems & Information Technologies/KYUSHU. He is currently an Associate Professor with the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. His research interests include system-level design methodologies for embedded systems, computer-aided designed of VLSI systems, and compilers for embedded systems. He has served on the organizing and program committees of several premier conferences including ASP-DAC and ICCAD. He is a member of ACM, IEEE, and IPSJ.

Hiroaki Takada is a Professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He received his Ph.D. degree in Information Science from University of Tokyo in 1996. He was a Research Associate at University of Tokyo from 1989 to 1997, and was an Assistant Professor and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IPSJ, and JSSST.