논문 99-24-6B-5 한국통신학회논문지 '99-6 Vol.24 No.6B

부정확 타스크의 최대가중치 오류를 최소화 시키는 온라인 스케쥴링 알고리즘

정회원 이 춘 희*, 류 원**, 송 기 현***, 최 경 희***, 정 기 현***, 박 승 규***

On-line Scheduling Algorithms for Reducing the Largest Weighted Error Incurred by Imprecise Tasks

Chun-Hi Lee*,Won Ryu**, Ki-Hyun Song**, Kyung-Hee Choi***, Gyhyun Jung***, Seung-Kyu Park*** *Regular Members*

요 약

본 논문은 단일프로세서 환경에서 부정확 타스크들을 선점형태로 스케쥴할 때 발생하는 오류의 최대치를 최소화하는 온라인 스케쥴링 알고리즘을 제시한다. 알고리즘은 상위수준 및 하위수준 알고리즘으로 나눈다. 상위수준 알고리즘에서는 매번 타스크가 도착할 때마다 스케쥴링을 수행하며, 대상 타스크들의 전체 오류 및 최대오류가 동시에 최소가 되도록 각 타스크들의 수행시간을 구한다. 하위수준 알고리즘에서는 배당된 타스크들의 할당 시간을 실제 프로세서가 어떤 순서대로 수행할지를 정한다. 상위수준 알고리즘은 주어진 조건을 LP 문제로 정형화하여 복잡도가 상당히 개선된 알고리즘을 제시하였고, 하위수준 알고리즘은 Shih and Liu[4] 의 온라인 알고리즘을 확장하였다. 시뮬레이션을 통하여 다수 경우의 상황을 실험한 결과 전의 연구방법 보다 상당히 개선된 효과, 즉, 최대오류를 더 최소화 시키는 것을 관찰하였다.

ABSTRACT

This paper proposes on-line scheduling algorithms that reduce the largest weighted error incurred by preemptive imprecise tasks running on a single processor system. The first one is a two-level algorithm. The top-level scheduling, which is executed whenever a new task arrives, determines the processing times to be allotted to tasks in such a way to minimize maximum weighted error as well as to minimize total error. The lower-level algorithm actually allocates the processor to the tasks. The second algorithm extends the on-line algorithm studied by Shih and Liu[4] by formalizing the top-level algorithm mathematically. The numerical simulation shows that the proposed algorithm outperforms the previous works in the sense that it greatly reduces the largest weighted error.

I. Introduction

The imprecise computation technique inherently prevents imprecise systems from being terminated abnormally due to a lack of processing time and produces non-optimal results with graceful degradation[1]. This allows users to get the approximate result with acceptable quality even when the exact result with the desired quality cannot be obtained in time. If all imprecise tasks are known before applying scheduling algorithms, it is enough to schedule them with static or off-line scheduling techniques such as ones proposed in [1,2,3,5].

* 아주대학교 컴퓨터공학과(chlee@seri.re.kr), ** 대전보건대학 경영정보과,
*** 아주대학교 정보및컴퓨터공학부({khchoi,khchung,sparky}@madang.ajou.ac.kr)
논문번호: 99027-0121, 접수일자: 1999년 1월 21일

However when on-line tasks arrive randomly, it is impossible to schedule the tasks at an instant and thus one should consider on-line scheduling algorithms for scheduling the tasks.

In an imprecise computation model, each task t_i is decomposed into two subtasks: the mandatory subtask M_i and the optional subtask O_i. Let M_i and O_i be the processing time of M_i and O_i, respectively, and $m_i + o_i = p_i$. In [4], Shih and Liu have proposed an on-line scheduling algorithm for minimizing the total error incurred by imprecise tasks. By the error e_i, we mean the difference between pi and the total amount of time assigned to the task, x_i, i.e., $e_i = p_i - x_i$. By weight w_i, we mean the importance ratio of error, and the weighted error is defined as $w_i e_i$. The total error is the sum of errors of all tasks, i.e., $\sum_{i=1}^{N} e_i$. By minimizing total error, the processor could be utilized at maximum. But, in some cases, *minimizing maximum weighted error is preferred* to only minimizing total error. Inspired by this philosophy, many off-line scheduling algorithms for minimizing maximum weighted error of imprecise tasks are proposed [3,5]. However little work has been done for on-line scheduling algorithms reducing the largest weighted error for preemptive imprecise tasks.

In this paper, we propose two on-line scheduling algorithms for imprecise on-line tasks. First, we address a two-level on-line scheduling algorithm that tries to reduce the largest weighted error : top-level and lower-level scheduling algorithm. The top-level scheduling algorithm is executed whenever a new task arrives, and determines the amount of processing times to be allotted to all schedulable tasks at that instant, while minimizing the maximum weighted error. The minimization process is modeled as a LP problem. Thanks to the model, the computation complexity of the procedure for allocating processing time to the scheduled tasks decreases significantly. For the lower-level scheduling algorithm that actually allocates the processing power to tasks, there are many well-known algorithms such as the algorithm by Shih and Liu [4] and the EDF algorithm[6]. As shown later, numerical simulations show that the distribution of weighted errors depends heavily not only on the top-level scheduling but also on the lower-level policy. For example, the lower-level scheduling based on a reservation policy [4] produces a schedule which utilizes maximum processing power. However it forces some tasks not to be scheduled, while they are scheduled under the EDF strategy.

This paper is organized as follows. Section two describes two proposed scheduling algorithms with mathematical proofs. In section three, the results of numerical simulations are presented, and the last section concludes this paper.

Ⅱ. Scheduling Algorithms

1. Two-level scheduling

It is a difficult problem to get an on-line schedule for imprecise tasks which satisfies criteria such as minimizing total error or minimizing maximum error. It is almost impossible to get an optimal on-line schedule for imprecise tasks, minimizing total error or minimizing maximum weighted error without a priori knowledge of the parameters for tasks such as arrival times and deadlines. In this section, we propose a heuristic two-level scheduling policy under the constraint that minimizes the largest weighted error.

The top-level schedule, which is executed whenever a new task arrives, determines the amount of processing times to be allocated to all schedulable tasks at that instant, i.e. all tasks present in the system. Therefore the top-level scheduling algorithm can be modeled as a static linear optimization problem: Given a set of preemptive imprecise tasks with identical arrival times, known deadlines, processing times of mandatory and optional parts, and weights, determine the amount of time to be allocated to each task so as not only to minimize the *maximum weighted error but also to minimize the* total error. Sub-section 2.1.1 describes the

top-level scheduling algorithm and its mathematical foundation in detail.

The lower-level algorithm actually schedules tasks into service. The maximum of service time is bounded by that obtained by the top-level algorithm. The amount of execution time that each task receives actually depends on the arrival time of new tasks. In other words, the amount of service time determined by the top-level algorithm is valid only until the next new task's arrival.

1.1 Top-level scheduling Formulation of problem

As described above, top-level scheduling algorithm distributes intervals to tasks present in the system, which allows us to assume that task arrival times are identical momentarily. Consider a set of N imprecise tasks with identical arrival times, deadlines $d_i \le d_{i+1}$, and arbitrary weights wi's. Let e_i be the error of task t_i and $x_{i,j}$ be the amount of time assigned to task t_i in interval (d_{j-1}, d_j). Then, the top-level algorithm may be formulated as :

minimize z

subject to

1) $z \ge wiei$, $i = 1,2,..., N$

2) $e_i = p_i - x_i$, $i = 1,2,..., N$

3) $\sum_{j=1}^{i} x_{i,j} = x_i$, $i = 1,2,..., N$

4) $\sum_{i=j}^{N} x_{i,j} = d_j - d_{j-1}$, $j = 1,2,..., N$

5) $x_i \ge m_i$, $i = 1,2,..., N$

The above formulation is a LP (Linear Programming) problem. This LP problem can be solved by iteration, solving the auxiliary optimization problem P_k, $k = N, N\text{-}1,...,2,1$. P_k is a sub-problem with tasks $t_k, t_{k+1},..., t_N$. Obviously P_1 is equivalent to the original problem. And the problem P_k can be written as :

minimize z

subject to

1) $z \ge wiei^* - wixi,k$, $i = k,k+1,...,N$

2) $\sum_{i=k}^{N} xi,k = dk - dk\text{-}1$

3) $xi,k \le ei^*$, $i = k,k+1,...,N$

4) $xi,k \ge 0$, $z \ge 0$, $i = k,k+1,...,N$

where e_i^* is the updated error of task t_i which is obtained by iteration during from interval (d_{N-1}, d_N) to interval (d_k, d_{k+1}). Note that all $x_{i,j}$, $j > k$, are known in the problem P_k, and it suffices to get $x_{i,k}$ for $i = k,k+1,k+2,...,N$.

In order to simplify the explanation without loss of generality, suppose that k is equal to one and let y_i be $x_{i,k}$. We introduce slack variables S_i for each task t_i, so that the formulation of the previous section becomes

minimize z

subject to

1) $z + wiyi - si = wiei^*$, $i = 1,2,...,N$

2) $\sum_{i=1}^{N} yi = l$

3) $yi \ge 0$, $si \ge 0$, $z \ge 0$, $i = 1,2,...,N$,

where l is the length of interval, (d_0, d_1), to be distributed to tasks.

The formulation can be rewritten as the following matrix form.

$$\begin{bmatrix} 1 & w1 & 0 & 0 & 0 & \cdots & 0 & -1 & 0 & 0\cdots0 \\ 1 & 0 & w2 & 0 & 0 & \cdots & 0 & 0 & -1 & 0\cdots0 \\ 1 & 0 & 0 & w3 & 0 & \cdots & 0 & 0 & 0 & -1\cdots0 \\ \cdots & & & & & & & & \cdots & \\ \cdots & & & & & & & & \cdots & \\ 1 & w1 & 0 & 0 & 0 & \cdots & 0 & -1 & 0 & 0\cdots0 \\ 1 & w1 & 0 & 0 & 0 & \cdots & 0 & -1 & 0 & 0\cdots0 \end{bmatrix} \begin{bmatrix} z \\ y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_N \\ s_1 \\ \cdot \\ \cdot \\ s_N \end{bmatrix} = \begin{bmatrix} w_1 e_1^* \\ w_2 e_2^* \\ \cdot \\ \cdot \\ \cdot \\ w_N e_N^* \\ l \end{bmatrix}$$

Let **B** be the left half of the first matrix (N+1 by N+1). And let y be the uppe r half of the vector in the left (N+1 elements) and lower one be s. Let E be the vector in the right side. Then, we have the following equation.

$$\begin{bmatrix} B & \begin{matrix} -I \\ 0 \end{matrix} \end{bmatrix} \begin{bmatrix} y \\ s \end{bmatrix} = E \qquad (1)$$

$$By - \begin{bmatrix} I \\ 0 \end{bmatrix} s = E$$

Let $\gamma_i = 1/w_i$ and $\gamma = 1/\sum \gamma_i$. Then, for matrix **B**, inverse matrix $\mathbf{B}^{-1}$ exists and,

$$B=\begin{bmatrix} 1 & w_1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & w_2 & 0 & \cdots & 0 \\ 1 & 0 & 0 & w_3 & \cdots & 0 \\ & \cdots & & & & \\ & \cdots & & & & \\ & \cdots & & & & \\ 0 & 0 & 0 & 0 & \cdots & w_N \\ 0 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \quad B\text{-}1=\begin{bmatrix} \gamma_1\gamma & \gamma_2\gamma & \cdots & \gamma_N\gamma & -\gamma \\ \gamma_1-\gamma_1\gamma_1\gamma & -\gamma_1\gamma_2\gamma & \cdots & -\gamma_1\gamma_N\gamma & \gamma_1\gamma \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ -\gamma_N\gamma_1\gamma & -\gamma_N\gamma_2\gamma & \cdots & \gamma_N-\gamma_N\gamma_N\gamma & \gamma_N\gamma \end{bmatrix}$$

Applying **B**-1 to the equation (1), we get

$$y\text{-}B\text{-}1\begin{bmatrix} I \\ 0 \end{bmatrix} s = B\text{-}1E$$

This equation becomes:

$$\begin{bmatrix} z-\gamma\sum\gamma_i s_i \\ y_1-\gamma_1 s_1+\gamma_1\gamma\sum\gamma_i s_i \\ y_2-\gamma_2 s_2+\gamma_2\gamma\sum\gamma_i s_i \\ \cdot \\ \cdot \\ \cdot \\ y_N-\gamma_N s_N+\gamma_N\gamma\sum\gamma_i s_i \end{bmatrix} = \begin{bmatrix} \gamma(\sum e_i^* - l) \\ e_1^* - \gamma_1\gamma(\sum e_i^* - l) \\ e_2^* - \gamma_2\gamma(\sum e_i^* - l) \\ \cdot \\ \cdot \\ \cdot \\ e_N^* - \gamma_N\gamma(\sum e_i^* - l) \end{bmatrix} \quad (2)$$

Let $\alpha = \gamma(\sum e_i^* - l)$ and $\beta = \gamma\sum\gamma_i s_i$. Then, equation (2) is rewritten as:

$$\begin{bmatrix} z-\beta \\ y_1-\gamma_1(s_1-\beta) \\ y_2-\gamma_2(s_2-\beta) \\ \cdots \\ y_N-\gamma_N(s_N-\beta) \end{bmatrix} = \begin{bmatrix} \alpha \\ e_1^*-\gamma_1\alpha \\ e_2^*-\gamma_2\alpha \\ \cdots \\ e_N^*-\gamma_N\alpha \end{bmatrix} \quad (3)$$

In the problem of LP, the optimal solution is found from the extreme point. Applying this property, we get the following theorems.

Theorem 1: *For some i, if* $e_i^* - \gamma_i\alpha \le 0$, *then* $y_i = 0$.

Proof: This comes from the fact that $e_i^* - \gamma_i\alpha = y_i - \gamma_i(s_i - \beta) \le 0$ means either $y_i = 0$ or $s_i = 0$ since the optimal solution can be found from the extreme point. This leads to $y_i = 0$.

Theorem 2: *Suppose that tasks are sorted in descending order of* $w_i e_i$. $e_k^* - \gamma_k^*\alpha_k < 0$ implies $e_{k+1}^* - \gamma_{k+1}^*\alpha_{k+1} < 0$.

Proof: It can be proved without difficulties, and so it is omitted.

With the help of the above properties, in the process of finding the optimal *yi* for a given interval, it is sufficient to perform the optimization process only for the tasks such that $e_i^* - \gamma_i\alpha \ge 0$. Once all tasks with $e_i^* - \gamma_i\alpha \ge 0$ have been found, the maximum weighted error and processing times to be allocated to the tasks are determined.

Theorem 3: *For all i, if* $e_i^* - \gamma_i\alpha > 0$, *then we have the solution* $z = \alpha$ *and* $y_i = e_i^* - \gamma_i\alpha$.

Proof: The optimal solution can be found from the extreme point in LP problem. At the extreme point where all *si* are equal to zero, the inequality relation 1) in the problem formulation becomes that of equality. This means that no more optimal solutions exist. The value β becomes (defined as the summation of *si*) equal to zero. From equation (3), we get the solution $y_i = e_i^* - \gamma_i\alpha$.

Based on the above theorems, the top-level algorithm is derived as in Figure 1. In the algorithm, u and v denote the indices of tasks and intervals, respectively, and initially all y_i's are set to zero. *l* denotes the length of interval.

Example.

Consider a task set with three imprecise tasks, t_1, t_2 and t_3. Their characteristics are shown in Table 1. r_i in Table 1 denotes the arrival time of task t_i. To schedule them, the proposed top-level algorithm is applied backwards to intervals (11,16), (8,11) and (3,8). Since t_3 is the only task that can be scheduled in the interval (11,16), the interval (11,16) is allocated only to t_3. However, in the next interval (8,11), as the first step, two time units are allocated to t_2, which is the processing time requirement of its mandatory part and then the optimization procedure is performed. The optimization procedure produces an outcome that $e_3^* = 2$, $e_2^* = 1$, $w_3 e_3^* = 0.82$, $w_2 e_2^* = 0.53$, $l = 1$,

$$\alpha = \alpha 2 = \gamma_2^*(e_2^* + e_3^* - l) = (1/(\frac{1}{0.41} + \frac{1}{0.53})$$

$(2+1-1)=0.231*2=0.46$,

$x_{32}= e_3^* - \gamma_3 a = 2-1/0.41*0.46 = 0.872$, and $x_{22}= e_2^* - \gamma_2 a = 1-1/0.53*0.46 = 0.128$. t_3 and t_2 have the same weighted error, 0.462. For the last interval (3,8), three unit times are allocated for task t_1, and then the remaining part of the interval is optimized. By performing the similar optimization procedure, we have that $e_3^*=1.128$, $e_2^*=0.872$, $e_1^*=2$, $w_3e_3^* = w_2e_2^* = 0.462$, $w_1e_1^*=0.06$, $l=2$, $\alpha = \alpha_3 = \gamma_3^*(\sum_{h=1}^{3} e_h^* - l)$

$= (1/(\frac{1}{0.41}+\frac{1}{0.53}+\frac{1}{0.03})(4-2)=0.02655$

L = a list of tasks sorted in decreasing value of w_ie_i, initially ∅
u = N
v = N
l = d_v-d_{v-1}
while(v≥1 and u≥1)
  if(m_u-x_u)>l, that is, task t_u has not yet received a sufficient amount of time to execute its mandatory part and the interval is sufficient lafge to allocate to t_u then
    x_u = x_u + l, v = v - l, l = d_v - d_{v-1}
  else if(m_u - x_u) = l then
    x_u = x_u + l, insert t_u into L
    u = u - l, v= v-l, l= d_v - d_{v-1}
  else if u>v then
    x_u = x_{u+1}, insert t_u into L
    u = u - 1,
  *else /*we are sure that u = v and all tasks t_u, t_{u+1}, ... t_N have received sufficient amount of time to execute their mandatory parts */*
  1) perform optimization procedure as follows
    (1) find the largest number k among k such that*
    e_{k+1} - $\gamma_{k+1}a_{k+1} \le 0$ and e_k* - $\gamma_k a_k$ > 0,*
    where a_k = γ_k($\sum_{h=1}^{k} e_h$* - l), γ_k* = $\sum_{h=1}^{k} \gamma_h$*
    (If e_k- $\gamma_k a_k$ > 0 for all k, then k* becomes the index of the last task)*
    (2) Let a be the a_k, and y_i = e_i* - $\gamma_i a$ for $i \le k$**
    (3) Update e_i by e_i* - y_i and x_i by for x_i+y_i the tasks t_i that have received additional processing times*
    (4) Update L by letting tasks with the maximum weighted error be one task
  2) u = u - 1, v= v-1
  end if
end while

Fig 1. Proposed algorithm producing minimum of maximum weighted error.

Table 1. Characteristics of a task set with three tasks

task	*ri*	*di*	*mi*	*oi*	*pi*	*wi*
t1	3	8	3	2	5	0.03
t2	3	11	2	1	3	0.53
t3	3	16	4	3	7	0.41

Table 2. Results of the top-level algorithm

task \ Interval	(3,8)	(8,11)	(11,16)	*Sum*
t1	3.230			3.230
t2	0.772	2.128		2.900
t3	0.998	0.872	5	6.870
Sum	5	3	5	13

*2=0.053, $x_{31}= e_3^* - \gamma_3 a = 1.128 - 1/0.41$ *0.0531=0.998, and $x_{21}= e_2^* - \gamma_2 a = 0.872 - 1/0.53*0.0531 = 0.7718$, and $x_{11}= e_1^* - \gamma_1 a = 2 - 1/0.03*0.0531 = 0.23$. As a result, the tasks t_1, t_2, and t_3 have different amount of errors, 1.77, 0.100, and 0.13, respectively. But they have the same weighted error,0.053. Table 2 shows the processing times allocated to the tasks in each interval by the top-level algorithm.

The complexity of the proposed algorithm in Figure 1 is *O(NLogN)*. The tasks with the same weighted error can be handled, throughout iterations, as one task such that its weight is equal to $1/\sum(1/w_i)$ and its error is equal to the sum of errors of the tasks with the same weighted error. Therefore, the number of iterations that optimization procedure (the step 1) in Figure 1) is executed is bounded by *O(N)*, which implies that the worst case complexity is *O(NLogN)*.

1.2 Low-level Scheduling

The purpose of the lower-level scheduling is to actually assign a processor to tasks according to the amounts of processing time determined by the top-level algorithm. Whenever a new task arrives, the top-level algorithm is executed and the time allocation information is recomputed. Then the processing time of optional part is modified as

the difference between the time produced by the top-level algorithm and m_i,(i,.e., x_i - m_i), but m_i itself is not modified. The lower-level scheduling allocates the processor to the tasks based on the modified processing times. There are many ways to schedule the tasks, once their processing times are determined.

One way for the lower-level scheduling is to utilize the results obtained by the top-level algorithm. Figure 2 depicts a schedule for the task set in Table 1, assuming that no new task arrives. Let LA be the low-level scheduling algorithm based on this philosophy. Another way for the lower-level scheduling is an algorithm trying to complete the mandatory parts of all tasks as soon as possible, and then trying to assign the processor to the tasks up to the time units determined by the top-level algorithm. Let LB denote this method. Figure 3 depicts the schedule by LB. One may also consider applying an off-line scheduling algorithm that focuses on minimizing total error. But it is not appropriate since the use of off-line scheduling only for the low-level scheduling introduces an additional overhead without any gain.

Note that in Figure 3, the interval (15,16) becomes idle and the final weighted error is not equal to that produced by the top-level algorithm.

As shown in Figure 3, the LB may underutilize the processor power and the maximum weighted error incurred by tasks may be greater than that by the LA. From the point of view of minimizing total error or minimizing maximum weighted error, it looks like that the LA works better than the LB. However, the result of simulation shows that there is no big difference between the values of minimum of maximum weighted error produced by both the LA and the LB for the cases that the processor utilization is high.

The task set that can be scheduled by the LB is larger than that by the LA. In other word any task set that is schedulable by the traditional EDF is still schedulable by the LB. In the next section

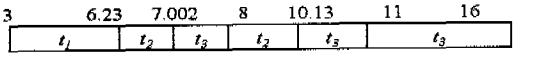


Fig. 2 Low-level scheduling (LA) using the information by the top-level schedule

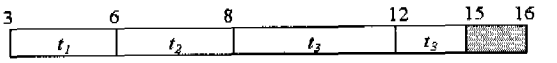


Fig. 3 Processing time allocation by the LB

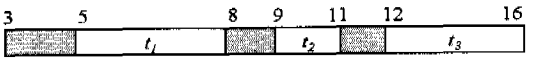


Fig. 4 Reservation when task t3 has arrived.

we describe another alternative, which is an extension of the algorithm proposed by Shih and Liu[4].

2. On-line Algorithm with Reservation(OAR)

In this section, we describe a technique called *on-line algorithm with reservation (OAR)* which is an extension of the on-line algorithm studied by Shih and Liu [4]. When a new task arrives, the OAR reserves an appropriate amount of intervals in a manner that the sum of reserved intervals is equal to the processing time for the mandatory part of the task. As in the algorithm by Shih and Liu, if it is not possible that the OAR reserves sufficient time for the task,then the OAR determines that the task is not schedulable. T he intervals that are not assigned for themandatory part are used for minimizing the maximum weighted error. For example, consider the following scenario. Two tasks, t_1 and t_2, have arrived before time 3, and their mandatory parts have been executed partially. At time 3, t_3 arrives. Figure 4 depicts a reservation in this case. The shaded rectangles represent the intervals available for reducing the maximum weighted error.

It is not difficult to see that the proposed top-level algorithm working for the intervals constructed from deadlines is still valid for the intervals (3,5), (8,9), and (11,12), as well. In fact, the allocation in Table 3 is same as the allocation

produced by the optimization procedure of the top-level algorithm. The total amount of time that is additionally allocated to the tasks t_1, t_2, t_3 in each interval is same as in Table 2.

Table 3. Allocated times by the OAR for the task set in Table 1.

Interval / task	(3,5)	(8,9)	(11,12)	Sum
t_1	0.230			0.230
t_2	0.772	0.128		0.900
t_3	0.998	0.872	1	2.870
Sum	2	1	1	

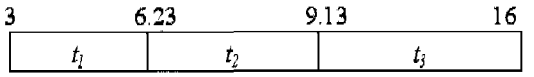


Fig. 5 Task allocation without a new task

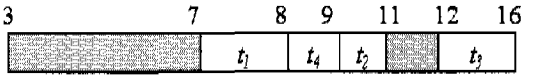


Fig. 6 Task allocation and reservation for a newly arrived task t4

Once the information is obtained as in Table 3, t_1 that has the earliest deadline is scheduled until a new task arrives or it terminates. If no task arrives during 3.23 time units (the sum of m_1 and the additional allocation), the interval (5,8) which has been reserved for t_1 is released. Next, the scheduler schedules t_2 during 2.900 time units. The same procedure is repeated until no more intervals are available. Figure 5 depicts a schedule for the case that no task arrives until time 18 and Figure 6 depicts a reservation by the OAR for the case that a new task t_4 with d_4 = 10 and m_1 = 1 arrives at time 5. In this case, the reservation for the mandatory part or the processing time allocation for minimizing maximum weighted error can be done in a similar way.

Figure 7 describes the OAR algorithm. It is executed whenever a new task arrives. Since it is a natural extension of the algorithm by Shih and Liu [4], it is easy to check that the algorithm OAR tries not only to minimize maximum weighted error but also to minimize total error. The weighted error incurred by tasks is not optimal in the sense that maximum weighted error may be greater than those by the off-line algorithms. And the maximum weighted error depends also on the arrival of tasks or the characteristics such as their processing times or weights. We claim a following theorem without proof, since it is a direct consequence from [4].

```
Let τ be a newly arrived task
Reserve intervals for t enough to execute its mandatory p
If the reservation fails, then
    the task t could not be scheduled by the proposed
    algorithm, and reject t
else
    (1) get a list of intervals, L, that could be used fo
    minimizing weighted errors.
    (2) perform the same minimization procedure for the
    intervals in L as in section 2.1.2.
    (3) modify the processing time according to the resul
    the optimization.
end if
Schedule tasks in increasing order of deadlines until a ne
task arrives, and modify the reservation as in the algorith
Shih and Liu[4]
```

Fig. 7 Algorithm OAR

Theorem 4.

The OAR produces schedules minimizing total error.

Ⅲ. Simulation

In order to check the performance of the proposed on-line algorithms, we have performed a series of experiments. For each experiment, we have generated a task set with three hundred tasks, modeled as an M/M/Infinity queuing system, in which the distribution characteristic of task arrival time is Poisson, the service time is exponentially distributed and there are infinite number of servers. The processing time of mandatory part of each task is taken uniformly from zero to (its deadline its release time) * p, where p is fixed arbitrarily from 0.2 to 0.9 for each experiment. The arrival rate over the service rate, (defined as) is the average number of tasks in system. As ρ or p become larger, the load of

processor also becomes higher. If the generated task set is not schedulable by the on-line algorithm by Shih and Liu, it is rejected, and regenerated.

Once task sets are generated and determined as schedulable by the proposed on-line algorithm, it is scheduled by the algorithm by Shih and Liu and by the proposed algorithm OAR, and then the weighted errors of all tasks are analyzed. Figure 8.a, 8.b,8.c and 8.d depict the distributions of weighted errors of the tasks in the task sets for ρ = 1,2, p =0.2, 0.4, and arbitrary weights. The x-axis represents the index of task ordered by arriving times, and the y-axis their weighted error. From the figures, we can easily note that the proposed algorithm outperforms the algorithm by Shih and Liu at the cost of manipulating the list of intervals that are not reserved for mandatory parts and calculating minimum of maximum weighted error.

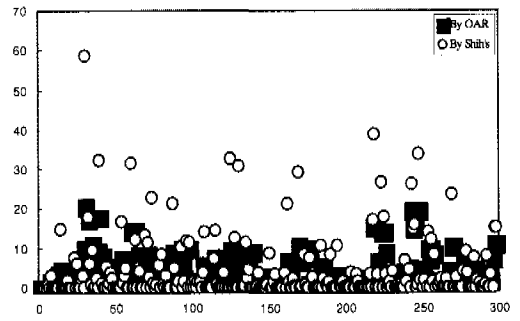


Fig 8. a Distribution of weighted error when ρ =1, p=0.2

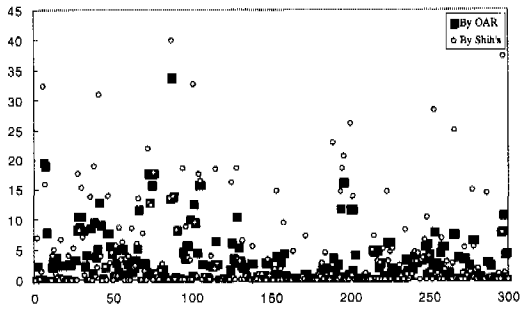


Fig. 8 b Distribution of weighted error when ρ =1, p=0.4

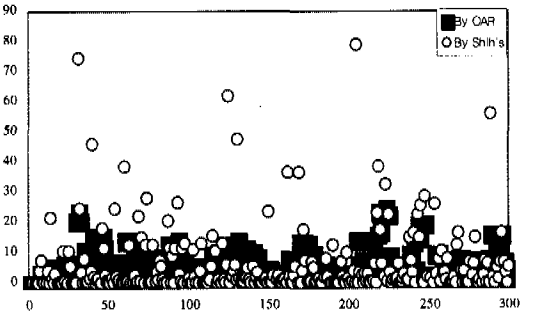


Fig. 8 c Distribution of weighted error when ρ =2, p=0.2

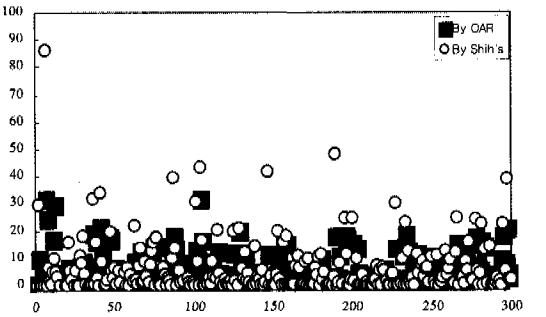


Fig. 8 d Distribution of weighted error when ρ =2, p=0.4

Fig. 8 Distribution of weighted errors with four different values of ρ and p.

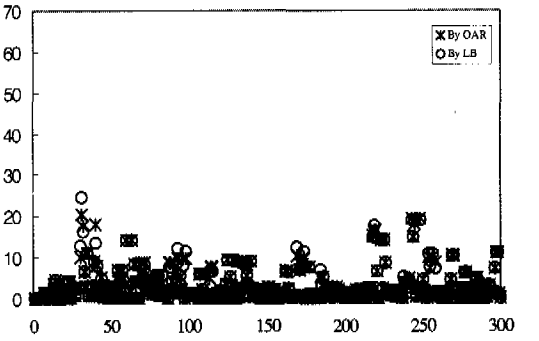


Fig. 9 Distribution of weghted errors with four different values of ρ and p.

Figure 9 depicts the weighted errors incurred by the same task set as that in Figure 8.a. Theoretically, the OAR has to produce smaller errors than the LB does. However the simulation

shows that they produce the almost same error distribution as shown in the figure 9. This result means that the distribution of processing times of mandatory parts and weights directly influences the distribution of error. The distribution of weighted error may change if the distributions for processing times of mandatory parts or weights are assumed differently.

Ⅳ. Conclusion

We presented a fast on-line algorithm for reducing largest weighted error where tasks have arbitrary weights. We transformed the optimization problem to a LP problem. With the help of valuable properties, the proposed algorithm rules out some tasks at the very beginning stage of scheduling and reduces the number of tasks actually to be scheduled. Without a great loss of timing complexity, weighted errors could be reduced significantly by the proposed algorithm. The proposed on-line algorithm could be useful for multimedia applications with QoS requirements.

References

[1] J.W.S.Liu, J.K.Lin, W.K.Shih, and A.C.Yu, Algorithms for Scheduling Imprecise Computations, Foundation of Real-Time Computing edited by A.M.Tilborg, Kluwer Academic Pub., pp.203-249,1991.

[2] W.K.Shih, J.W.S.Liu, and J.Y.Chung, Fast Algorithms for Scheduling Imprecise Computations, SIAM J. on Computing, vol.20, no.3, July 1991.

[3] W.K.Shih, and J.W.S.Liu, Algorithms for Scheduling Imprecise Computations with Timing Constraints to Minimize Maximum Error, IEEE Tr. On Computers, vol.44, no.3, March 1995.

[4] W.K.Shih, and J.W.S.Liu, On-line Scheduling of Imprecise Computations to Minimize Error, Proc.of the 13th Real-Time Systems Symposium, Phoenix, Arizona, pp.280-289,December 1992.

[5] K.Ho,J.Leung and W.Wei, Minimizing Maximum Weighted Error for Imprecise Computation Tasks, J. Algorithms, Vol.16, pp.431-452, 1994.

[6] J.A.Stankovic,M.Spuri,M.D.Natale, and G. Buttazzo, Implication of Classical Scheduling Results For Real-Time Systems, IEEE Computers, Vol.28, No.6, pp.16-25, June 1995.

이 춘 희(Chun-Hee Lee) 정회원

1964년 2월 : 성균관대학교 영어영문학과 졸업

1985년 2월 : 연세대학교 전산학과 석사

1968년 3월~1998년 12월 : 시스템공학연구소 및 전자통신연구소 책임연구원

1990년 3월~현재 : 아주대학교 컴퓨터공학과 박사과정

류 원(Won Ryu) 정회원

1983년 2월 : 부산대학교 계산통계학과 졸업

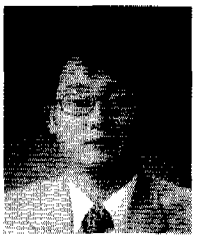
1988년 2월 : 서울대학교 대학원 계산통계학과 석사

1998년 8월 : 성균관대학교 대학원 정보공학과 박사과정 수료

1989년 8월~현재 : 한국전자통신연구원 통신처리팀 선임연구원 재직중

<주관심 분야> 데이터통신시스템, 병렬처리 시스템, 이동컴퓨팅

송 기 현(Ki-Hyun Song) 정회원



1985년 2월 : 충남대학교 계산통계학과 졸업

1987년 2월 : 충남대학교 계산통계학과 석사

1999년 2월 : 아주대학교 컴퓨터공학과 박사

1990년 3월~현재 : 대전보건대학 경영정보과 조교수

<주관심 분야> 실시간스케쥴링, 자연어처리

최 경 희(Kyung-Hee Choi) 정회원
1976년 2월 : 서울대학교 사범대 수학과 졸업
1979년 : 프랑스 그랑제꼴 ENSEEIHT 공학석사
1982년 : 프랑스 Paul Sabatier 대 공학박사
1982년 3월~현재 : 아주대학교 정보및컴퓨터공학부 교수
<주관심 분야> 운영체제, 분산시스템, 실시간시스템, 프로그래밍방법론

정 기 현(Gyhyun Jung) 정회원
1984년2월 : 서강대학교 전자공학과 졸업
1988년 : University of Illinois, EECS 공학석사
1990년 : Purdue University 공학박사
1992년 3월~현재 : 아주대학교 전자공학부 교수
<주관심 분야> 멀티미디어, VLSI, 실시간시스템

박 승 규(Seung-Kyu Park) 정회원
1974년 2월 : 서울대학교 공대 응용수학과 졸업
1976년 2월 : KIAST 전산과 석사
1982년 3월 : 프랑스 INP de Grenoble 공학박사
1992년 3월~현재 : 아주대학교 정보및컴퓨터공학부 교수
<주관심 분야> 멀티미디어시스템, 실시간시스템, 이동컴퓨팅