소프트 실시간 시스템을 위한 두 단계 스케쥴링 알고리즘

김 재 훈†

요 약

본 논문은 소프트 실시간 시스템을 위한 스케쥴링을 간단히 하기 위하여 "*두 단계 데드라인*" 방법을 제안하였다. 각각의 작업은 두 개의 데드라인을 갖는데, 첫 번째 데드라인과 두 번째 데드라인이라 부른다. 첫 번째 데드라인은 일반 실시간 시스템의 데드라인과 같다. 두 번째 데드라인은 첫 번째 데드라인 보다 늦은 시각인데, 늦은 결과가 아직은 사용할 만한 가장 늦은 시각이다. 첫 번째 데드라인 이내에 끝난 일은 만점을 주지만, 첫 번째 데드라인을 놓치고 두 번째 데드라인 이내에 끝난 일에 대해서는 부분 점수만 인정한다. 본 논문에서, 첫 번째 데드라인, 두 번째 데드라인, 부분 점수 등을 고려하여 다이나믹하게 일들의 우선 순위를 정하는 방법을 고안하였다. 시뮬레이션 결과를 통하여 두 단계 스케쥴링 알고리즘이 소프트 실시간 시스템과 일시적으로 과부하가 걸린 하드 실시간을 처리하는데 적절한 방법임을 확인하였다.

Two-Level Scheduling for Soft Real-Time Systems

Jai-Hoon Kim†

ABSTRACT

This paper presents an algorithm for scheduling jobs in soft real-time systems. To simplify the scheduling for soft real-time systems, we introduce two-level deadline scheme. Each job in the system has two deadlines, which we call first-level and second-level deadlines, respectively. The first-level deadline is the same as the deadline in traditional real-time systems. The second-level deadline is later than the first-level deadline, and defines the latest point in time when the result is still acceptable. Partial-credit is given for jobs meeting the second-level deadline but missing the first-level deadline, whereas jobs meeting the latter are given full credit. We heuristically compute priorities of jobs in a dynamic way by combining the first-level and second-level deadlines with the partial-credit. Simulation results indicate that our two-level scheduling algorithm is a viable approach for dealing with both soft real-time systems and temporary overloaded hard real-time systems.

1. Introduction

In hard real-time systems, it is essential to meet deadlines. If a deadline is missed in a hard real-time system, the result is useless or may even cause disastrous consequences. For example, embedded systems and recovery procedures for highly available systems are in this category. On the other hand, in soft real-time systems (e.g., on-line transaction systems, telephone switches, stock price quotations, multimedia, etc.), meeting deadlines is desirable. How-

※ 본 연구는 아주대학교 1998년도 정착연구비 지원으로 진행되었음.
† 정 회 원 : 아주대학교 정보통신대학 정보 및 컴퓨터공학부 교수
논문접수 : 1998년 4월 29일, 심사완료 : 1998년 11월 18일

ever, results generated after the deadline may still be useful. The usefulness of each soft real-time job missing the deadline is a function of tardiness, which depends on a characteristic of the job.

This paper presents an algorithm for scheduling jobs in soft real-time systems. To simplify the scheduling for soft real-time systems, we introduce a *"two-level deadline"* scheme. Each job in the system has two deadlines, which we call *first-level* and *second-level* deadlines, respectively. The first-level deadline is the same as the deadline in traditional real-time systems. The second-level deadline is later than the first-level deadline, and defines the latest point in time when the result is still acceptable. The difference between the first-level and second-level deadlines is therefore equal to the maximum acceptable tardiness. *Partial-credit* is given for jobs meeting the second-level deadline but missing the first-level deadline, whereas jobs meeting the latter are given full credit.

We describe an adaptive scheduling algorithm to schedule jobs with first-level and second-level deadlines so as to maximize the total credit (or minimize the total penalty). The algorithm is based on EDF (Earliest-Deadline-First). We heuristically compute priorities of jobs in a dynamic way by combining the first-level and second-level deadlines and the partial-credit. As an extension, an adaptive scheduling algorithm also considers weight and expected execution time of each job to maximize the benefit. Simulation results indicate that our two-level scheduling algorithm is a viable approach for dealing with both soft real-time systems and temporary overload in hard real-time systems.

2. Related Work

The majority of real-time systems, even many real-time systems have been thought as hard real-time systems, categorized as soft real-time systems [4]. Many researches deal with soft real-time systems. Lee et al. [5] propose a scheduling algorithm for soft aperiodic tasks. Their objective is to schedule all periodic tasks and to obtain fast response time for aperiodic tasks. They use hybrid scheduling method combining fixed priority strategy for aperiodic tasks and deadlinewise preassignment for periodic tasks. Ripoll et al. [10] propose an optimal scheduling algorithm for soft aperiodic tasks. They transform a soft aperiodic task into a hard task by assigning a deadline. After the transform, aperiodic tasks are handled as hard periodic tasks. This algorithm is an optimal for periodic task set in terms of schedulability and provides the shortest response time for aperiodic tasks. Nagy and Bestavros [9] propose an admission control scheme and overload management technique for soft-deadline transactions. Their transaction model consists of two parts : a primary task and a compensating task. A transaction is considered to be finished by executing one of followings : the primary task is completed (successfully committed) with positive profit, or the compensating task is completed (safely terminated) with no profit. Only committed transactions are benefit to the system. Their objective is to maximize the benefit of primary tasks that finish by their deadlines. Buttazzo et al. [2] compare the performance of scheduling algorithms which use different priority and different guarantee mechanisms for an overloaded real-time systems. Imprecise computation [1] reduces the execution time of jobs by skipping an optional part of the job to avoid missing their deadlines if the result is acceptable. In our two-level deadline scheme, a deadline is extended, instead of reducing computation time if the tardiness is acceptable.

3. Adaptive Scheduling Algorithm

3.1 Basic Scheme

To characterize the soft real-time jobs, the following parameters are given for each job where N

is the number of preemptable jobs : $J_i(0 \le i \le 1)$,

- p_i : priority value.
- e_i : execution time.
- d_{1i} : the first-level deadline.
- d_{2i} $(d_{2i} \ge d_{1i})$: the second-level deadline.
- c_i $(0 \le c_i \le 1)$: given credit in meeting the second-level deadline after missing the first-level deadline.

Our adaptive scheduling algorithm is based on EDF (Earliest-Deadline-First), and schedules jobs with arbitrary release times and deadlines.

In addition to the deadline in the traditional sense (the first-level deadline), the second-level deadline and credit are also important factors in computing priorities in our two-level deadline scheme. We need to combine the two deadlines (the first-and the second-level deadlines), credit, and weights. We heuristically compute priorities at every time unit. (Note that in the following a job with a higher priority value has a lower priority. Deadlines are relative to current time.) :

- before the first-level deadline : $p_i(d_{1i}, d_{2i}, c_i) = d_{1i} + (d_{2i} - d_{1i})c_i$
- after the first-level deadline : $p_i(d_{2i}, c_i) = d_{2i}/c_i$

The motivation for so computing the priorities of jobs without weight is :

1. Preemptive EDF scheduling is used as the basis algorithm.
2. The extension of deadline is considered by using the second-level deadline (d_{2i}) and the credit c_i. When c_i=1, the second-level deadline is the same as the first-level deadline. (The first-deadline can be extended to the second-level deadline without any quality degradation.) On the other hand, when c_i=0, there is no benefit in meeting the second-level deadline once the first-level deadline has been missed. (The second-level deadline is useless.)
3. After missing the first-level deadline, only the second-level deadline (d_{2i}) and the credit (c_i) are used. Priority value increases (priority decreases) as c_i is low.

3.2 Parameter Tuning

For a precise parameter tuning for computing the priorities of jobs, two weight values (w_b and w_a) are considered. Two weight values have following meanings :

- w_b : this value is used for weighting the extension of deadlines and partial credits of jobs before the first-level deadline.
- w_a : this value is used for weighting the jobs after missing the first-level deadline before the second-level deadline.

We heuristically compute the priority of job with two weight parameters (w_b and w_a) as follows :

- before the first-level deadline : $p_i(d_{1i}, d_{2i}, c_i) = d_{1i} + w_b(d_{2i} - d_{1i})c_i$.
- after the first-level deadline : $p_i(d_{2i}, c_i) = w_a(d_{2i}/c_i)$.

We tuned the two weight values (w_b and w_a) by comparing the performance by varying one of the two weight values. Two system parameters are also used to vary the types of the systems in the simulation :

- softness factor (S) : This parameter denotes how long the second-level deadline can be extensible in maximum. For instance, S=4 denotes that the second-level deadline can be extended up to S-1=3 times as long as the minimum slack time (relative first-level deadline minus maximum execution time of jobs). The second-level deadline is uniformly distributed between the first-level deadline and the maximum bound of extension for the second-level deadline.

- Credit (c) : $(0 \le c_i \le 1)$, credit in meeting the second-level deadline after missing the first-level deadline.

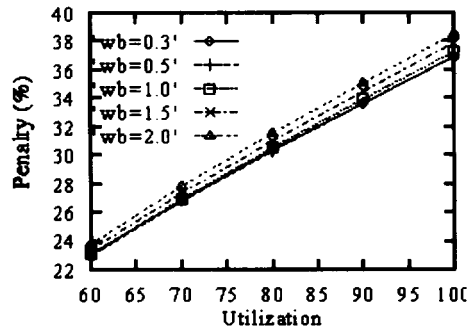
For the performance comparison, we define a performance metric, penalty, as follows :

$$penalty = \left(\sum_{j=1}^{M_1} (1 - c_{m1_j}) + M_2\right)/N,$$

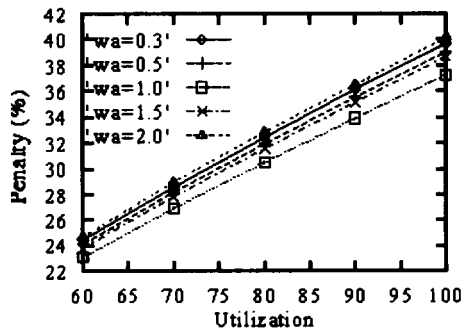
where M_1 is the number of jobs missing the first-level deadline but meeting the second-level deadline, c_{m1j} is the credit (c) of the j-th job missing the first-level deadline only, M_2 is the number of jobs missing the second-level deadline, and N is the number of total jobs released.

To tune the two weight values, we measure the performance by varying one weight value. First, we vary the weight w_b and use fixed value for the other weight (w_a) and other parameters : w_a=1.0, S=4, and c=0.6. The inter arrival time of jobs is Poisson distribution. (The average inter arrival time depends on utilization.) Execution times of jobs are uniformly distributed in interval [5,15], relative deadline is uniformly distributed in interval [15,20]. The experiments run for a total of 1,000,000 CPU time units. (Fig. 1) shows the performance (penalty). This figure suggests that the performance is optimal (suboptimal) when the value w_b is not greater than 1.0. We also measure the performance with other parameter values (e.g., S=2 and c=0.4). Appropriate weight values are similar to these.

As the next step, we measure performance by varying another weight value w_a. We use suboptimal value w_b=1.0 obtained from above simulation. The (Fig. 2) shows the performance, which suggests that w_a is optimal in near w_a=1.0. Simulation results with other parameter values (e.g., s=2 and c=0.4) are similar. Thus, we decide the weight parameters as w_b=1.0 and w_a=1.0. These values seem to be simple but might be reasonable values. These values (w_b=1.0 and w_a=1.0) are used for other simulations in this paper.



(그림 1) 품질비교 (변수 : w_b)
(Fig. 1) Quality Comparison (variable : w_b)

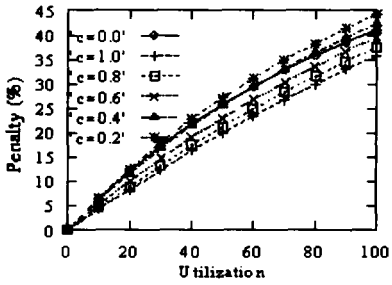


(그림 2) 품질비교 (변수 : w_a)
(Fig. 2) Quality Comparison (variable : w_a)

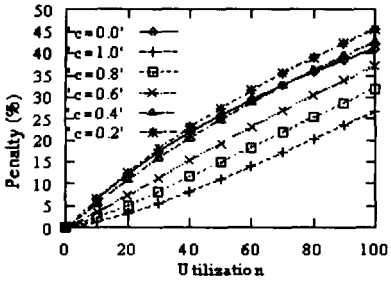
3.3 Performance Comparisons

(Fig. 3), (Fig. 4), and (Fig. 5) show the performance comparisons with the tuned weight values, w_b=1.0 and w_a=1.0. In these simulations, execution time of jobs is uniformly distributed in interval [5,15], and the first level (relative) deadline (d_1) is uniformly distributed in interval [15,20]. The experiments run for a total of 1,000,000 CPU time units. In these figures, "c=x" denotes that partial credit is x for all jobs. (In special, "c=0.0" denotes that the two-level deadline scheme is not used.) As the credit c increases the penalty of adaptive scheduling algorithm decreases just as we expected. However, there are no gains in an adaptive scheme as c decreases, especially in high utilization. The reason

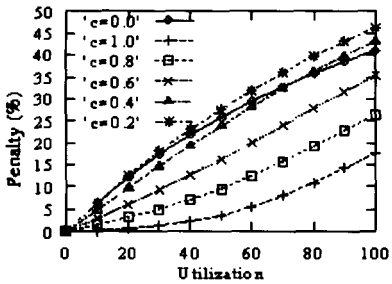
is that spending CPU time to meet the second-level deadline for the small partial credit will cause other jobs to miss the first level deadline (miss the full credit). Therefore, our adaptive algorithm takes more advantage as credit (c) increases. Another aspect of these results is that the difference of penalty between high and low credits increases as softness factor (S) increases. (Fig. 5) covers wider range of penalty than (Fig. 3) and (Fig. 4).



(그림 3) 품질비교 (S=2)
(Fig. 3) Quality Comparison (S=2)



(그림 4) 품질비교 (S=4)
(Fig. 4) Quality Comparison (S=4)



(그림 5) 품질비교 (S=8)
(Fig. 5) Quality Comparison (S=8)

3.4 Precision and Confidence Level

For our simulation, we use 1,000,000 time units, and the average execution time of job is 10 time units. Thus, 100,000 jobs are released on the average in 100% utilization and 10,000 jobs are released in 10% utilization. We can expect that the simulation results are very precise. In this subsection, we present the precision and the confidence level of our simulation results with the following parameters : the soft factor S=2 and the partial credit c=0.6. The overall point estimation is θ. $100(1-\alpha)\%$ confidence interval with the degree of freedom f is between $\theta - t_{\alpha/2,f}\sigma(\theta)$ and $\theta + t_{\alpha/2,f}\sigma(\theta)$. We perform this simulation 5 times. Thus, degree of freedom f=R-1=5-1=4. Table 1 shows estimation, precision (standard error), and $t_{\alpha/2,f}\ \sigma(\theta)$ of penalty of quality for each utilization.

〈표 1〉 정밀도 및 신뢰도
〈Table 1〉 Precision and Confidence Level Utilization

Utilization(%)	Estimation	Precision	$t_{\alpha/2,f}\ \sigma(\theta)$				
			99%	98%	95%	90%	80%
10	5.1096	0.07080	0.3257	0.2655	0.1968	0.1508	0.1083
20	9.8910	0.05444	0.2504	0.2041	0.1513	0.1159	0.0832
30	14.5014	0.06612	0.3041	0.2479	0.1838	0.1408	0.1011
40	18.8166	0.10810	0.4976	0.4056	0.3007	0.2304	0.1655
50	22.9068	0.06166	0.2836	0.2312	0.1714	0.1313	0.0943
60	26.6770	0.07620	0.3505	0.2857	0.2118	0.1623	0.1165
70	30.2278	0.06830	0.3141	0.2561	0.1898	0.1454	0.1045
80	33.5448	0.04492	0.2066	0.1684	0.1248	0.0956	0.0687
90	36.6060	0.04177	0.1921	0.1566	0.1161	0.0889	0.0639
100	39.4694	0.04679	0.2152	0.1754	0.1300	0.0996	0.0715

4. Extension of Adaptive Scheduling Algorithm

4.1 Adapt to Weighted Jobs

Now, we consider about extension of the previous basic scheme described in Section 3.1 where we assumed that all jobs have equal value. However, in the real world, different job may have different values. The value of jobs is often overlooked in real-time systems, many of which simply focus on deadlines rather than a combination of value and deadlines [11].

In this section, we assume that jobs may have different weights (values). By this assumption, we heuristically compute priorities for weighted jobs at every time unit. (Note that in the following a job with a higher priority value has a lower priority. Deadlines are relative to current time.) :

- before the first-level deadline :
  $p_i\ (d_{1i},\ d_{2i},\ c_i,\ w_i) = (d_{1i} + w_b(d_{2i} - d_{1i})c_i)(W + 1 - w_i)$.
- After the first-level deadline :
  $P_i\ (d_{2i},\ c_i,\ w_i) = w_a(d_{2i}/c_i)(W + 1 - w_i)$,

where W is the maximum weight of jobs and w_i is the weight of job J_i.

The motivation in computing the priorities of weighted jobs is to give a job (J_i) the priority in proportional to the weight of job (w_i). If job J_i has a weight w_i, then the penalty can be computed as follows :

MR = (weight sum of jobs missing deadline)/(weight sum of all jobs)

$$= \sum_{j=1}^{M} W_{m_j} / \sum_{i=1}^{N} W_i,$$

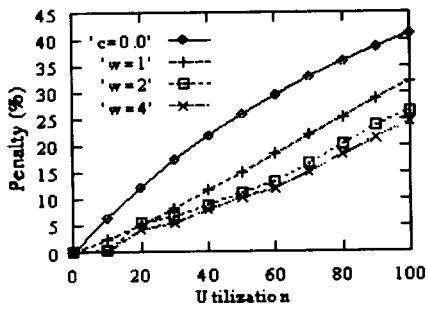
where M is the number of jobs missing the deadline, N is the number of jobs released, w_i is weight of job J_i, and w_{mj} is weight of the j-th job missing the deadline. Finally, if we combine with the two-level deadline, the miss ratio is computed as follows :

MR =

$$\left(\sum_{j=1}^{M_1} W_{m1_j}(1 - c_{m1j}) + \sum_{k=1}^{M_2} W_{m2_k}\right) / \sum_{i=1}^{N} W_i,$$

where M_1 is the number jobs missing the first level deadline only, M_2 is the number jobs missing the both deadlines, w_{m1j} is the weight of the j-th job missing the first-level deadline only, w_{m2k} is the weight of the k-th job missing the second-level deadline, and c_{m1j} is the credit of the j-th job missing the first-level deadline only.

(Fig. 6) shows that the new adaptive scheduling algorithm for weighted jobs decreases the penalty. In these experiments, execution times of jobs are uniformly distributed in interval [5,15], and the first-level deadline is uniformly distributed in interval [15,20]. The experiments run for a total of 1,000,000 time units. In this figure, "c=0.0" denotes that two-level deadline scheme is not used, and "w=x" denotes that weight is uniformly distributed between 1 and x. As we expected, the penalty decreases as w increases by giving higher priority to jobs with higher weight (i.e., more valuable jobs).



(그림 6) 품질비교 (가중치 작업 : S=4, c=0.6)
(Fig. 6) Quality Comparison (weighted jobs : S=4, c=0.6)

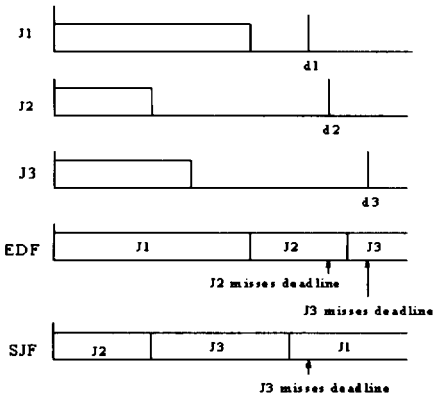
4.2 Adapt to Execution Time

In the above section, we do not count the execution time for computing the priority of job. In section 3, only the two-level deadline is considered, and, in the subsection 4.1, two-level deadline and weight of job are considered in computing the priority.

However, we have to consider the execution time as well as deadline and weight to reduce the penalty. (In order to simplify the problem, we assume that the weights of all jobs are identical.) As shown in (Fig. 7), job J1 has the earliest deadline, but has the longest execution time. On the other hand, job J2 and job J3 have later deadline, but have shorter execution times. (Fig. 7) shows an example where the SJF(Shortest-Job-First) scheduling algorithm has a lower miss ratio than the EDF (Earliest-Deadline-

First) scheduling algorithm.

From the above motivation, we consider both deadline and execution time in computing the priority of jobs. We assumed that the importance (value) of each job is same in this subsection 4.2. We consider two ways in assigning the priority of jobs with deadline and execution time. One way is weighting execution time on the previous priority (in Section 3.1) to assign the new priority, and the other way is weighting between deadline and execution time.



(그림 7) EDF와 SJF
(Fig. 7) EDF vs. SJF

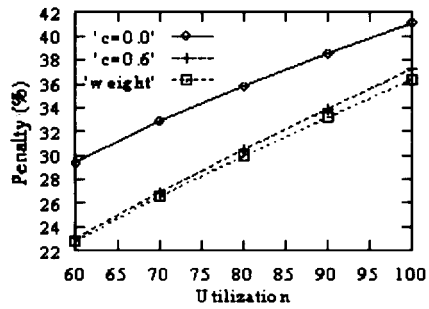
4.2.1 Weight on Execution Time

We weight the execution time on the previous priority of job (in Section 3.1). Thus, a job with a shorter execution time has a higher priority. (Note, again, that a job with a larger priority value has a lower priority. Deadlines are relative to current time.) :

- before the first-level deadline :
  $p_i(d_{1i}, d_{2i}, c_i, e_i) = (d_{1i} + w_b (d_{2i} - d_{1i}) c_i) e_i$
- after the first-level deadline :
  $p_i(d_{2i}, c_i, e_i) = w_a (d_{2i}/c_i) e_i$,

where e_i is the execution time job J_i.

We measure the performance with the same parameters of jobs as the previous simulation. (Fig. 8) shows that this scheme can improve performance (a little though) at the overloaded system.



(그림 8) 품질비교 (S=4, c=0.6)
(Fig. 8) Performance Comparison (S=4 and c=0.6)

4.2.2 Weight between Deadline and Execution Time

We also weight between deadline and execution time to compute the priority of jobs as follows :

$$p_i(d_i, e_i) = R\ |d_i| + (1 - R)\ |e_i|,$$

where $|d_i|$ is a normalized deadline, $|e_i|$ is a normalized execution time, and R is another weight parameter between 0 and 1 to decide which part is more weighted (when R=1, only deadline is counted (EDF); when R=0 only execution time is computed (SJF)) :

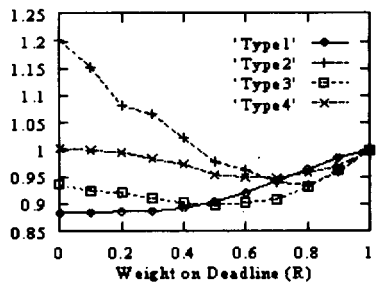
- before the first-level deadline :
  $p_i(d_{1i}, d_{2i}, c_i, e_i) = R\ |d_{1i} + (d_{2i} - d_{1i})c_i| + (1 - R)\ |e_i|.$
- after the first-level deadline :
  $p_i(d_{2i}, c_i, e_i) = R\ |d_{2i}/c_i| + (1 - R)\ |e_i|.$

An appropriate compromise between deadlines and execution times (i.e., choosing appropriate value R) of jobs in computing the priority can reduce the penalty. However, it is hard to estimate an optimal value of R (it depends on the job's parameters).

To study the effect of R, we compare the penalty by varying the jobs parameters. For performance comparison, we assume that a set of consecutive jobs, called a "burst", are released at consecutive time clicks,

and an interval between two bursts (difference in release time of two jobs, first released job in a burst and the last released job in the previous burst) is constant (t_s). The number of jobs in a burst is exponentially distributed (average is n_s). For the experiment, we vary two parameters, t_s and n_s. (Fig. 9) shows the relative miss ratio. In these experiments, all jobs are assumed to have the same softness factor, S=2, and the same partial credit, c=0.8. We use n_s=9 and t_s=180 for "type1"; n_s=4 and t_s=200 for "type2"; n_s=4 and t_s=100 for "type3"; and n_s=2 and t_s=45 for "type4". Execution time (e) and relative deadline (D) are uniformly distributed on (2,10) and (11,18) for "type1", (5,10) and (11,40) for "type2", (3,10) and (11,20) for "type3", and (1,10) and (11,15) for "type4", respectively. The experiments run for a total of 50,000 CPU time units.

In general, to minimize the average miss ratio, choosing R near the middle, on (0.5, 0.7), is reasonable in many cases. This result suggests that EDF scheduling may not be an appropriate algorithm except the case of existing feasible scheduling. The compromising scheme that gives weight between deadline and execution time for computing priority of job can improve performance (reduce penalty).



(그림 9) 상대적 품질비교 (s=2, c=0.8)
(Fig. 9) Relative Quality Comparison (S=2 and c=0.8)

5. Conclusion and Future Work

We present an adaptive scheduling algorithm for soft real-time systems based on a two-level deadline scheme. In this scheme, an adaptive scheduling algorithm tries to meet the first-level deadline for each job. If it fails, then this algorithm tries to meet the second-level deadline with some penalty. Simulation results show that this scheme can reduce the penalty. (We define the penalty for the two-level deadline scheme). As the first extension, we also consider a scheduling algorithm for weighted jobs, which gives a higher priority for more valuable jobs. In the second extension, the execution time of jobs is considered in computing the job's priority. Simulation results show that the penalty is decreased further with this extended adaptive scheduling algorithm.

To simplify the discussion, we consider two-level deadlines. More than two levels can be also considered for the future work.

References

[1] P. Alexander et al., "Managing Transient Overload in an Imprecise Computation Systems," Proc. of IEEE Workshop on Imprecise and Approximation Computation, pp.1-5, Dec. 1992.

[2] G. Buttazzo, M. Spuri, and F. Sensini, "Value vs. Deadline Scheduling in Overload Conditions," Proc. of Real-Time Systems Symposium, pp. 90-99, Dec. 1995.

[3] M. Garey et al., "Scheduling Unit-Time Tasks with Arbitrary Release Times and Deadlines," SIAM Journal on Computing, Vol.10, No.2, pp. 256-269, May 1981.

[4] E. D. Jensen, "Eliminating the hard/soft real-time dichotomy," Computer & Control Engineering Journal, pp.15-20, Feb. 1997.

[5] J. Lee, S. Lee, and H. Kim, "Scheduling Soft Aperiodic Tasks in Adaptable Fixed-Priority Systems," Operating Systems Review, Vol.30, No.4, pp.17-28, Aug. 1996.

[6] J. Lehoczky et al., "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," Proc. of the IEEE 1987 Real-Time Systems Symposium, pp.261-270, Dec. 1987.

[7] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," Journal of ACM, Vol.20, No.1, pp. 46-61, Jan. 1973.

[8] J. Liu, Real-Time Systems (draft).

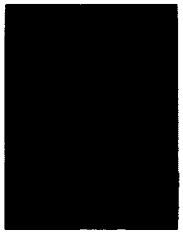
[9] S. Nagy and A. Bestavros, "Admission Control for Soft-Deadline Transactions in ACCORD," Proc. of Real-Time Technology and Applications, pp.160-165, Jun. 1997.

[10] I. Ripoll, A. Crespo, A. Garcia-Forness, "An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Dynamic-Priority Preemptive Systems," IEEE Trans. on Software Engineering, Vol.12, No.6, pp.388-400, Jun. 1997.

[11] M. Spuri, et al., "Implications of Classical Scheduling Results for Real-Time Systems," IEEE Computer, Vol.28, pp.16-25, 1995.

[12] J. Stankovic, "Misconceptions about Real-Time Computing," IEEE Computer, pp.10-19, Oct. 1988.

[13] J. Strosnider et al., "The deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," IEEE Transactions on Computers, Vol.44, No.1, pp.73-91, Jan. 1995.



김 재 훈

e-mail : jaikim@madang.ajou.ac.kr

1984년 서울대학교 제어계측공학과 (학사)

1993년 Indiana University, Computer Science(석사)

1997년 Texas A&M University, Computer Science(공학박사)

1998년~현재 아주대학교 정보 및 컴퓨터공학부 조교수

관심분야 : 분산시스템, 실시간시스템