다중처리기 시스템에서 데드라인과 여유시간을 통합한 실시간 스케줄링 기법

(Integrating Deadline with Laxity for Real-time Scheduling in Multiprocessor Systems)

조 성 제 †

(Seongje Cho)

요 약 실시간 시스템에서 많은 요청을 처리하기 위해 다중처리기 구조가 필수적이 되었다. EDF나 LLA와 같은 기존의 실시간 온라인 스케줄링 알고리즘들은 다중처리기 시스템에서 실시간 태스크들을 스케줄링하는 데 적합하지 않다. EDF의 경우 문맥교환 오버헤드가 낮지만 다중처리기 이상현상을 보이며, LLA의 경우 준최적이지만 문맥교환 오버헤드가 높다. EDZL은 두 알고리즘의 문제점을 일부 해결하였으나 세 개 이상의 처리기에서는 준최적이 아니다. EDA2는 과부하 단계에서 좋은 성능을 보이지만 준최적이 아니다. 본 논문에서는 새로운 두개의 온라인 스케줄링 알고리즘 ED/LL과 ED2/LL을 제안한다. ED/LL은 다중처리기에서 준최적이며, 정상부하 단계에서 낮은 문맥교환 오버헤드와 높은 성공률을 보인다. 그러나, 시스템이 과부하 상태가 되면 ED/LL은 비효율적이다. 이를 해결하기 위해, ED2/LL은 정상부하 단계에서는 ED/LL또는 EDZL을 사용하고 과부하 단계에서는 EDA2를 사용한다. 실험을 통해 ED2/LL이 정상부하 단계에서는 물론 과부하 단계에서도 좋은 성능을 보임을 확인할 수 있었다.

키워드 : 실시간 스케줄링, 다중처리기, ED/LL (Earliest Deadline / Least Laxity), ED2/LL

Abstract For real-time systems, multiprocessor support is indispensable to handle the large number of requests. Existing real-time on-line scheduling algorithms such as Earliest Deadline First Algorithm (EDF) and Least Laxity Algorithm (LLA) may not be suitable for scheduling real-time tasks in multiprocessor systems. Although EDF has low context switching overhead, it suffers from "multiple processor anomalies." LLA has been shown as suboptimal, but has the potential for higher context switching overhead. Earliest Deadline Zero Laxity (EDZL) solved somewhat the problems of those algorithms, however is suboptimal for only two processors. Another algorithm EDA2 shows very good performance in overload phase, however, is not suboptimal for multiprocessors. We propose two on-line scheduling algorithms, Earliest Deadline/Least Laxity (ED/LL) and ED2/LL. ED/LL is suboptimal for multiprocessors, and has low context switching overhead and low deadline miss rate in normal load phase. However, ED/LL is ineffective when the system is overloaded. To solve this problem, ED2/LL uses ED/LL or EDZL in normal load phase and uses EDA2 in overload phase. Experimental results show that ED2/LL achieves good performance in overload phase as well as in normal load phase.

Key words : real-time scheduling, multiprocessor, ED/LL (Earliest Deadline / Least Laxity), ED2/LL

1. Introduction

As Internet becomes part of our daily life, web-based systems must be able to provide multimedia support to handle the rich data types in many web applications. Multimedia tasks must meet their application deadlines, or the output is considered a low quality (e.g. unstable video playback) or im-

*이 연구는 2002학년도 단국대학교 대학연구비의 지원으로 연구되었음.

† 정 회 원 : 단국대학교 정보컴퓨터학부 컴퓨터과학전공 교수
sjcho@dku.edu

논문접수 : 2002년 3월 18일
심사완료 : 2002년 10월 14일

possible to use (e.g. incorrect sound pitch). To guarantee a better real-time performance, many systems have utilized more computing resources (CPU) to handle a higher workload. Multiprocessor architecture thus has become very common for web servers. However, real-time scheduling on multiprocessor systems is a challenging problem. A real-time system with two processors may not produce a performance twice as good as a single processor system. In this paper, we study the problem of on-line scheduling algorithms for real-time tasks on multiprocessor systems, and provide a good solution to the problem.

In real-time systems where the release times of tasks are sporadic, there may be time intervals during which it is impossible to meet all task deadlines due to critical or exceptional conditions. A system is said to be in the *overload phase* during an interval I, if it is impossible to meet all the deadlines of ready tasks during the period I. Otherwise, it is said to be in the *normal load phase*. Because of the stochastic nature of the release times of sporadic tasks, it is impractical to eliminate overload situations in large and dynamic systems by means of system or application design methods[1]. Therefore, designing effective scheduling algorithms for overload phases as well as for normal load phases is important to the construction of realistic real-time systems.

Any scheduling decision will affect the set of ready tasks in the near future, since it determines which tasks are executed and which tasks are not. A current scheduling decision of executing less urgent tasks may cause unnecessary overload phase in the future. Therefore, finding an effective scheduling algorithm for normal-load phase is critical to the overall success of real-time scheduling. When a system goes into an overload phase, a different goal should be pursued: more important tasks should be executed first to handle the emergency rather than trying to meet the deadlines of all tasks[1,2,3,4].

Recently, there has been an increasing demand for multiprocessor support in the systems like as web server systems. However, few satisfactory on-line scheduling algorithms for real-time tasks in multiprocessor systems have been found. Two well-known algorithms, *Earliest Deadline First Algorithm* (EDF)[5] and *Least Laxity Algorithm* (LLA)[6, 7], have been shown to be optimal in uniprocessor systems. In multiprocessor systems, EDF has a low context switching overhead, but it can produce arbitrarily low processor utilization since that deadline miss ratio is high under EDF[4,8]. LLA is shown to be suboptimal (see section 2.2 for the definition of suboptimality), but it is impractical due to high context switching overhead in multiprocessor systems[9,10]. To solve the problems of EDF and LLA algorithms, *Earliest Deadline Zero Laxity* (EDZL) algorithm has been proposed to support real-time scheduling on multiprocessors [11]. EDZL has advantages from both EDF and LLA. It schedules tasks with both deadline and laxity values of tasks. The EDZL has low context switching overhead and low deadline miss rate. But it is not suboptimal for three or more processors, and does not show good performance in overload phase[11]. Another algorithm EDA2[11] schedules tasks based on the deadline, and examines the laxity value of every task as time advances. If there is any task with negative laxity, EDA2 removes the task from system. EDA2 shows very good performance for multiprocessors when the system is overloaded, even though it is not suboptimal.

In this paper, we first present *Earliest Deadline/Least Laxity* (ED/LL) algorithm which solves the problems of existing algorithms. Like as EDZL, ED/LL is also a hybrid algorithm of EDF and LLA. We will show that ED/LL is suboptimal for multiprocessors like LLA. ED/LL has good performance in meeting the deadlines of tasks, while the number of context switches is smaller than LLA. However, it is not effective in overload phase. To solve the problem, we also propose an adaptive real-time scheduling algorithm *ED2/LL* that uses ED/LL or EDZL in normal load phase, but uses EDA2 in overload phase. ED2/LL shows good performance irrespective of the system load level.

This paper is organized as follows. In section 2, we describe existing on-line scheduling algorithms and

task model. Sections 3 and 4 present the new algorithms ED/LL and ED2/LL respectively. Section 5 evaluates the proposed algorithms and compares with the existing ones. In section 6, we conclude our work.

2. Background

2.1 Related Work

There has been much research on on-line scheduling algorithm based on EDF. Chetto[12] studied the localization and duration of idle time interval in EDF, and proposed the algorithm that utilizes the idle time duration to schedule hard real-time sporadic tasks. Jeffay[13] proposed an optimal algorithm that schedules tasks with shared resources on one processor, based on EDF. Baruah[2,3] derived an upper bound that can be derived an upper bound that can be achieved when tasks with different value densities are scheduled on one processor in overload phase, and proposed an algorithm D*, an extended on-line algorithm from EDF. D* has similar performance with *Best Effort* (BE) algorithm proposed by Locke[4], but guarantees to achieve lower bound on any condition.

Baruah et al. have achieved some positive result in real-time multiprocessor scheduling[14,15,16]. Dertouzos[6] analyzed the performance of EDF and LLA on multiprocessors. Anand and Sanjoy[8] proposed an EDF-based multiprocessor algorithm for scheduling periodic task systems, and proved that the algorithm schedules any periodic task system with utilization ($m^2/(2m-1)$ on m identical processors. At the same time, they pointed out that EDF is not optimal on multiprocessors. As discussed briefly in Section 2.3, Cho et al. [11] presented two real-time scheduling algorithms EDZL and EDA2 on multiprocessors. EDZL is a suboptimal scheduling algorithm for two processors, and EDA2 provides a good overload handler. In this paper, we propose two scheduling algorithms that can substitute for LLA and EDZL in multiprocessor systems.

2.2 Task Model and Suboptimality

A *ready task* is defined as a task which has been released before or at the current time, and is yet to be completed, while a *future task* is a task which has not yet been released. A ready task running on a processor is called a *current task*. A schedule S is said to be *valid* for a set of tasks if and only if every task deadline is met in S. A set of tasks is said to be *feasible on m processors* if and only if there exists a valid schedule. A set of tasks is said to be *schedulable under a scheduling algorithm Q*, if Q can generate a valid schedule for the set of tasks.

A scheduling algorithm is said to be *optimal* if every feasible set of tasks (both ready and future tasks) is schedulable under Q. A scheduling algorithm is said to be *suboptimal* if every feasible set of ready tasks is schedulable under Q. The difference between optimality and suboptimality is whether future tasks are considered or not. One major assumption of on-line scheduling systems is that the release times of tasks are not available beforehand. This assumption forces the scheduler to make a decision based on the ready tasks, not based upon the prediction of what tasks will be released in the near future. This brings us to the suboptimality of a scheduling algorithm as a criterion.

Our system model depends on the following assumptions:

1. each task in the task set is aperiodic,
2. each task consists of only one job,
3. the deadline and the computation time of a task are given when the task is released,
4. the tasks do not communicate or synchronize with each other, and
5. the tasks can be preempted at any point in the computation.

To schedule a task set meeting these requirements, we use a priority, preemptive scheduler. A task τ_i is represented by a two-tuple (c, d), where c is the remaining computation time required to complete τ_i and d is the deadline relative to the current time. The computation time and deadline for τ_i vary as τ_i is executed and as time advances. The laxity of a task $\tau_i=(c, d)$ is defined by $d-c$. The laxity value of a task is often viewed as a measure of urgency. A negative laxity indicates that a deadline will be missed. Note that the deadline values of ready tasks are independent of scheduling decisions, while the

laxity values of the tasks are changed by scheduling decisions.

2.3 EDZL and EDA2 Algorithms

EDZL algorithm [11] is shown in Figure 1. EDZL first checks if there is an urgent task (i.e. task with zero laxity). If every task has a positive laxity, EDZL is performed in the same way as EDF. When a task with zero laxity occurs, however, it is dispatched promptly by preempting the task with positive laxity among current tasks. If the number of the processors is larger than the number of the zero laxity tasks, EDZL then tries to dispatch the tasks with earlier deadline to the processors that are not assigned to the tasks with zero laxity. That is, there may be the case under EDZL that some processors execute the tasks with zero laxity and the other processors execute the tasks with a positive laxity and an earlier deadline. If the number of tasks with zero laxity is larger than the number of processors, the system removes some of the tasks that cannot be dispatched. As we can know from [11], EDZL is suboptimal on two processors, but not suboptimal on more than two processors. For example, consider a set of ready tasks {(2,2), (2,2), (4,5), (4,5). (1,4)}, which is schedulable on three processors under LLA. Under EDZL, the set of tasks is not feasible: $\{(2,2), (2,2), (4,5), (4,5). (1,4)\}_0 \rightarrow \{(1,1), (1,1), (4,4), (4,4)\}_1 \rightarrow \{(3,3), (4,3)\}_2$ ... The system is in overload phase at the time t_1. Another disadvantage is that EDZL is not suitable for the overload phase.

```
Algorithm EDZL

1 while TRUE do
2    while there is any ready task with zero laxity in the queue do
3       if the laxity values of all the current tasks are zero then
4             remove the ready task from the system
5       else
6             preempt the current task with positive laxity and
                    dispatch the ready task with zero laxity
7       endif
8    endwhile

9    if there are i CPUs assigned to the tasks with positive laxity then
10            dispatch i tasks with earlier deadline to the i CPUs
11   endif
12   wait for next schedule event     // the current tasks are running
13 endwhile
```

Fig 1 *Earliest Deadline Zero Laxity* Algorithm (EDZL)

```
Algorithm EDA2

1 While TRUE do
2   while there is a task with negative laxity do
3         remove it from the system
4         endwhile
5         dispatch m tasks with earlier deadline to all m-CPUs
6         wait for next schedule event
                // the current tasks are running
7 endwhile
```

Fig 2 *EDA2* algorithm

Another algorithm EDA2 achieves a good performance in overload phase [11]. Figure 2 shows the description of EDA2 algorithm. Under EDF, a task with negative laxity value may remain in the system because EDF considers only the deadline of each task but does not consider the laxity value. So, the task to fail finally may waste processor time, which could degrade the performance of the overall system. (See table 1 and 2.) To solve the problem, EDA2 examines the laxity value of each task at every scheduling point, and removes immediately the tasks with negative laxity from the system. EDA2 then executes at any time as many as possible up to m tasks whose deadline is the closest for m-processor systems. Even though there is a task with zero laxity, the task with the closest deadline has the highest priority under EDA2 while the zero laxity task has the highest priority under EDZL. EDA2 is the same as EDF except that EDA2 can remove the tasks with a negative laxity at each scheduling point. Like EDF, the shortcoming of EDA2 is not optimal on multiprocessors either as shown in Table 1. The set of ready tasks {(5,7), (4,6), (7,9)}, which is feasible under LLA and EDZL, cannot be scheduled by EDF and EDA2 on two processors.

Generally EDA2 can achieve a good performance in overload phase under multiprocessors while EDZL and LLA show better performance in normal phase. Table 2 shows the scheduling stages of the set of tasks {(2,3), (3,5), (2,4), (4,4), (3,3), (4,4), (2,5)} at time t_0 on three processors. The number of tasks that meet their deadlines is 3 under EDF, 4 under LLA and EDZL, and 5 under EDA2, respectively. EDZL and LLA show also comparative performance because they remove the task with non-positive laxity which

Table 1 A scheduling example on two processors

time	LLA	EDZL	EDF	EDA2
t_0	(5,7) (4,6) (7,9)			
t_1	(4,6)(3,5)(7,8)	(4,6)(3,5)(7,8)	(4,6)(3,5)(7,8)	(4,6)(3,5)(7,8)
t_2	(3,5)(3,4)(6,7)	(3,5)(2,4)(7,7)	(3,5)(2,4)(7,7)	(3,5)(2,4)(7,7)
t_3	(3,4)(2,3)(5,6)	(3,4)(1,3)(6,6)	(2,4)(1,3)(7,6)	(2,4)(1,3)(7,6)
t_4	(3,3)(1,2)(4,5)	(3,3)(0,2)(5,5)	(1,3)(0,2)(7,5)	(1,3)(0,2) Fail
t_5	(2,2)(0,1)(4,4)	(2,2) (4,4)	(0,2) (6,4)	(0,2)
t_6	(1,1) (3,3)	(3,3)	(5,3)	

Table 2 A scheduling example on three processors

time	***EDF***	***EDA2***	***LLA*** and ***EDZL***
t_0	(2,3) (3,5) (2,4) (4,4) (3,3) (4,4) (2,5)	(2,3) (3,5) (2,4) (4,4) (3,3) (4,4) (2,5)	(2,3) (3,5) (2,4) (4,4) (3,3) (4,4) (2,5)
t1	(1,2) (3,4) (1,3) (4,3) (2,2) (4,3) (2,4)	(1,2) (3,4) (1,3) (4,3) (2,2) (4,3) (2,4)	(2,2) (3,4) (2,3) (3,3) (2,2) (3,3) (2,4)
t2	Succ (3,3) succ (4,2) (1,1) (4,2) (2,3)	succ (3,3) succ Fail (1,1) Fail (2,3)	Fail (3,3) (2,2) (2,2) (1,1) (2,2) (2,3)
t3	(3,2) (3,1) succ (3,1) (2,2)	(2,2) succ (1,2)	Fail Fail (1,1) succ (1,1) (2,2)
t4	(2,1) (2,0) (2,0) (2,1)	(1,1) succ	succ succ (1,1)
t5	(1,0) Fail Fail (1,0)	Succ	succ
t6	Fail Fail		

cannot be dispatched just in time. EDF shows the worst performance in overload phase on multi-processors since it schedules the tasks whose deadline will be missed and wastes the processor time on them.

3. ED/LL Algorithm

From section 2, we know that deadline or laxity alone cannot make an effective scheduling algorithm. Based on this observation, we propose an algorithm, *Earliest Deadline/Least Laxity* (ED/LL) that uses both factors. Figure 3 shows ED/LL algorithm.

ED/LL is performed in the same way as EDF for all tasks if there is no urgent task with zero laxity. If a task with zero laxity occurs, however, ED/LL is performed in the same way as LLA. Whenever the number of the tasks with zero laxity is larger than 0 for m-processor systems, ED/LL executes as many as possible up to m tasks which have the smallest laxity by preempting the current tasks with larger laxity. ED/LL differs from EDZL in this point. Remember that EDZL dispatches not only the tasks with zero laxity but also the tasks with earlier deadline and positive laxity if the number of the tasks with zero laxity is smaller than the number of processors. If ties occur among laxity values of the current tasks when ED/LL uses LLA, the task with the earliest deadline is chosen first. When all tasks with zero laxity have been completed, ED/LL switches scheduling algorithm back to EDF from LLA.

```
Algorithm ED/LL

1 while TRUE do
2       while there is any task with negative laxity do
3             remove it from the system
4       endwhile

5       if there is any task with zero laxity then
6             dispatch m tasks with smaller laxity to all m-CPUs
7             if there is any task with zero laxity which is
                    not dispatched then
8                   remove it from the system
9             endif
10      else if all tasks have a positive laxity then
11            dispatch m tasks with earlier deadline to all m-CPUs
12      endif
13      wait for next schedule event
              // the current tasks are running
14 endwhile
```

Fig 3 *ED/LL* Algorithm

For example, consider a set of ready tasks {(5,7), (4,7), (7,9)}, which is not schedulable on three processors under EDF and EDA2. As we can see from Table 3, the set of tasks is feasible under LLA, EDZL, and ED/LL. It is easy to see that ED/LL can successfully schedule all task sets that are schedulable under strict EDF, since ED/LL is actually an EDF algorithm until any task with zero laxity occurs. Note that any task scheduled by its zero laxity will never be preempted by other tasks. Therefore, ED/LL has lower context switching overhead than that of LLA. In Table 3, there are two preemptions at time 3 and 5, and five context switches under ED/LL, while there are three preemptions at time 3, 4 and 6, and six context switches under LLA.

Table 3 A scheduling example on three processors

	LLA	EDZL	ED/LL
t_0	(5,7) (4,7) (7,9)		
t_1	(4,6)(4,6)(6,8)	(4,6)(3,6)(7,8)	(4,6)(3,6)(7,8)
t_2	(3,5)(4,5)(5,7)	(3,5)(2,5)(7,7)	(3,5)(2,5)(7,7)
t_3	(3,4)(3,4)(4,6)	(2,4)(2,4)(6,6)	(2,4)(2,4)(6,6)
t_4	(2,3)(2,3)(4,5)	(1,3)(2,3)(5,5)	(1,3)(2,3)(5,5)
t_5	(1,2)(1,2)(4,4)	(0,2)(2,2)(4,4)	(1,2)(1,2)(4,4)
t_6	(1,1)(0,1)(3,3)	(1,1)(3,3)	(1,1)(0,1)(3,3)

The implementation cost of ED/LL may be higher than that of LLA since ED/LL keeps two ready queues: a *deadline queue* and a *laxity queue*. Every ready task is arranged in the order of deadline on the deadline queue and in the order of laxity value on the laxity queue. Each task under ED/LL is on two queues at the same time, while each task under LLA is on the laxity queue alone. Therefore the cost of queue management under ED/LL is almost twice that of LLA. However, ED/LL may be more desirable than LLA considering the overhead of preemptions and context switches. If the number of context switches decreases, the costs of managing queues as well as the number of saving and restoring task contexts are reduced. We will now show that ED/LL is suboptimal on multiprocessors.

Lemma 1: For a given scheduling algorithm Q and a set of ready tasks at time t_i, X, let *Sched(X, Q)* be the set of tasks which is feasible at time t_{i+1}. If X is feasible on m-processors and Q is suboptimal, *Sched(X, Q)* is always feasible on m-processors.

Proof: Trivial from the definitions of feasibility and suboptimality.

Theorem 1: ED/LL is a suboptimal algorithm on multiprocessors.

Proof: Let X be the set of ready tasks at time t_0, and X' be *Sched(X, ED/LL)*. We shall prove the theorem by contradiction, assuming that ED/LL is not suboptimal. Suppose that X is schedulable under LLA on m-processors but X' is not schedulable under ED/LL. There are two cases that must be considered. At time t_1,

1) The number of tasks with zero laxity is $m+k$ (there can be tasks with non-zero laxity),
2) The number of tasks with laxity less than zero is k

for some $k>0$. Without loss of the generality, we will consider only the minimum unschedulable set of tasks with $k=1$. We will show that a task set X which is schedulable under LLA cannot be scheduled by LLA if ED/LL is not suboptimal. The laxity value is represented by L.

(***case 1***) The number of tasks with zero laxity is $(m+1)$ at time t_1 under ED/LL.

There can be three kinds of tasks at time t_0 when scheduled by ED/LL.

a) p tasks with $L=0$, $c \geq 2$,
b) $(m+1)$ (p tasks with $L=1$, $c \geq 1$, $d \geq x$,
c) m(p tasks with $L>1$, $c \geq 1$, $d<x$

for $0 \leq p \leq m$ and $x>3$. If $p \geq 1$, ED/LL is suboptimal since ED/LL uses LLA. For the proof, we consider the smallest task set which is not schedulable under ED/LL when $p=0$. If $p=0$, we have $(m+1)$ tasks with $L=1$, $c=x-1$, $d=x$, and m tasks with $L=x-2$, $c=1$, $d=x-1$. These tasks require $(m+1)(x-1)+m$ processing time units in next x time units. But obviously, there are only mx processing time units available. The requirement $(m+1)(x-1)+m=(m+1)x-1$ is greater than mx because $x>3$.

Therefore, X is not schedulable under LLA, which contradicts the assumptions.

(***case 2***) A task τ_i has $L=-1$ at time t_1 under ED/LL.

When the task τ_i with L=0 is not executed at time t_0, τ_i has L= -1 at time t_1. Since τ_i is not scheduled at time t_0, the number of tasks with L=0 at time t_0 must be greater than or equal to m. That is, the number of tasks with L=0 at time t_0 is at least (m+1). Hence, case 2 is equivalent to case 1. We consider case 2 because this case can occur in EDF though it never happens in ED/LL.

By theorem 1, ED/LL can schedule all ready tasks that can be scheduled under LLA on multiprocessors. ED/LL can schedule the set of tasks X= {(2,2),(2,2),(4,5),(4,5),(1,4)} on three processors, while EDZL cannot. That is, X is schedulable under ED/LL like as LLA since *Sched(X, ED/LL)* = {(1,1),(1,1),(3,4),(4,4),(1,3)}, and *Sched(Sched(X, ED/LL), ED/LL)* = {(3,3),(3,3),(1,2)}. However, X is not schedulable under EDZL because *Sched(X, EDZL)* = {(1,1),(1,1),(4,4),(4,4)}, and *Sched(Sched(X, EDZL), EDZL)* = {(3,3),(4,3)}. Consequently, ED/LL shows better validity than EDZL.

4. Adaptive Algorithm: ED2/LL

Real-time systems may go into an overload phase at any time because of the stochastic nature of the release times of sporadic tasks. Therefore, it is important to handle an overload phase as well as a normal load phase. EDA2 shows a good performance in the overload phase, while LLA and ED/LL show better performance in the normal load phase. EDA2 shows best performance in a highly overload phase. This means that the scheduling algorithm based on both deadline and laxity is more efficient than an algorithm based on laxity or deadline alone under some situations. We saw from Table 2 that scheduler based on deadline performs well for a set of the tasks only if the tasks with negative laxity can be removed early.

To handle effectively normal load and overload phases together, we propose an adaptive algorithm ED2/LL that uses an efficient algorithm depending on system load condition. Figure 4 shows ED2/LL algorithm. ED2/LL first checks up the status of system load. If the system is normally loaded, ED2/LL uses EDZL or ED/LL; otherwise it uses

```
Algorithm ED2/LL

1 while TRUE do
2       if 'Utility of system >= UB then
3           perform EDA2
4       else
5           if the number of processors is less than 3 then
6               perform EDZL
7           else
8               perform ED/LL
9           end if
10      endif
11 endwhile
```

Fig 4 ED2/LL Algorithm

EDA2. In ED2/LL, the switching between algorithms occurs dynamically at run time according to system load. On multiprocessor, it is not easy that the scheduler determines exactly the state of system load. If rate monotonic algorithm is used for periodic tasks on uni-processor, we can predict whether a task set will meet all of its deadlines by comparing the utilization of the task set to a schedulable bound. For the rate monotonic priority assignment, a task set τ_1, τ_2, ..., τ_n is schedulable if

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \le n(2^{1/n} - 1)$$

where C_i is the computation time for a task of τ_i and P_i is the period for τ_i. The C_i/P_i is the utilization for τ_i.

Anand and Sanjoy proposed a utilization-based sufficient feasibility condition for scheduling periodic task systems on m unit-speed processors [8]. They also presented a definition that a periodic task system τ = {τ_1, τ_2, ..., τ_n} is said to be a *light system on **m** processors* if it satisfies the following two properties.

Property 1: $U(\tau) \le m^2/(2m-1)$

Property 2: For each $\tau_i \in \in$, $U_i \le m^2/(2m-1)$

where U is the utilization. Our goal in this paper is not to propose a new model for deciding overload phase for aperiodic tasks on multiprocessors, but to study an adaptive real-time algorithm. We can still use the utilization as the criteria for determining the system load condition, even though our target system is not a periodic real-time system.

The utilization for an aperiodic task set τ_1, τ_2, ..., τ_n can be computed as

$$U = \sum U_i = \sum_{i=1}^{n} C_i/(D_i - T_i)$$

where C_i is the remaining computation time, D_i is the deadline for the task τ_i, and T_i is current time. For example, we can determine that system is in the overload phase if U is larger than certain **utilization bound** (*UB*). We can see some experimental results of ED2/LL performed with some utilization bounds in the next section.

5. Performance Evaluation

In this section, we evaluate the performance of ED/LL and ED2/LL in comparison to EDF, EDA2, LLA, and EDZL by simulation. Each task is specified by three parameters: the worst case computation time, laxity (deadline), and the arrival time of the task. The start time of each task was generated according to Poisson distribution with (as the mean number of task arrival per unit time. The execution time was generated according to the normal distribution with mean 10 and the standard deviation 2. The deadline was also generated according to the normal distribution.

5.1Notation

To evaluate the performance of each algorithm, we have compared the percentage of tasks whose deadlines are not missed and the number of context switches under each algorithm. Table 4 shows symbols and their meaning described in this paper. The experiments were performed by varying the number of processors, the rate of task arrival, and the laxity values.

Table 4 Meaning of denotations

Symbol	Meaning
m	The number of processors
λ	The rate of task arrival
L	The mean laxity value
SR	The percentage of tasks that meet their deadlines
CS	The number of context switches
PE	The number of preemptions
UB	The utilization bound for ED2/LL

5.2 Effects of the Number of Processors

In this experiment, we use the following parameters: the value of λ = 0.5, L = 10, and the standard deviation of laxity value is 2. The number of tasks generated is 200 and the ***UB*** is 0.8. That is, ED2/LL uses EDZL or ED/LL if the utilization of the given task set is smaller than 0.8; otherwise it uses EDA2. Figure 5 shows the effects of *m* on SR and CS. *SR* increases with more processors. To schedule all tasks completely, LLA, EDZL, ED/LL, and ED2/LL need 8 or more processors, while EDF and EDA2 need 9 or more processors. The given set of tasks is schedulable under ED/LL and ED2/LL on 8-processors but not schedulable under EDA2. ED/LL and ED2/LL are better real-time schedulers than EDA2 since ED/LL and ED2/LL show the better validity in meeting the deadlines of tasks.

As we can see from Figure 5, EDF shows the lowest SR and LLA shows the largest CS. EDA2 shows smaller SR than ED/LL and ED2/LL in normal load phase, even though it achieves good performance in overload phase. Comparing with LLA in normal load phase, ED/LL and ED2/LL achieve almost the same SR, and much smaller CS. When the system is overloaded ($m \leq 6$), EDA2 shows the best performance while EDF shows poor performance. We can also see that the proposed algorithm ED2/LL performs very well not only in normal load phase but also in overload phase while ED/LL shows a little poor performance in overload phase. This means that the algorithm based on deadline may be efficient only if the tasks with negative laxity are removed early from system in some cases.

5.3 Effects of the Rate of Task Arrival

This experiment was performed with m = 5, L = 13, and the standard deviation of laxity value =2. ***UB*** is 0.8 and the number of tasks generated is 200. Figure 6 shows the effects of λ on SR and CS. When the rate of task arrivals is relatively large, the probability of deadline miss becomes large for all algorithms since the system may be overloaded. The effects of λ on SR are opposite to those of the number of processors on SR. The given set of tasks is feasible when $\lambda \leq 0.3$ under LLA, EDZL, ED/LL, and ED2/LL, while it is feasible when $\lambda \leq 0.2$ under EDF and EDA2. Moreover, since ED/LL and ED2/LL show

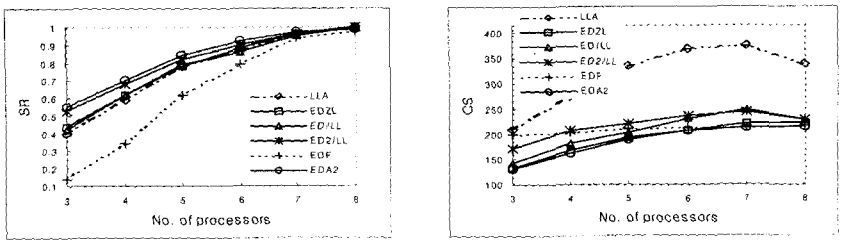

Fig 5 Effects of the number of processors (m)

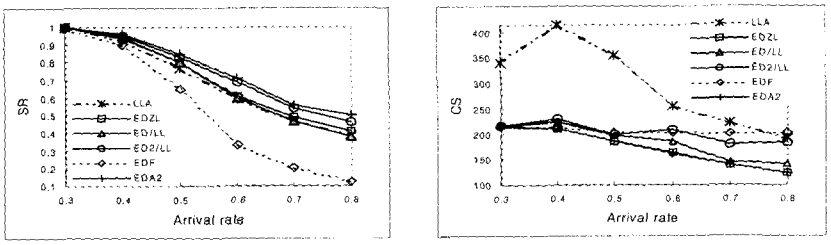

Fig 6 Effects of the rate of task arrival (λ)

much smaller CS than LLA when λ=0.3, they are better than LLA.

The number of context switches, especially for LLA, tends to decrease when $\lambda \leq 0.4$. This is because a large number of tasks that missed their deadlines are removed early from the ready queue, resulting in a small task set. The number of context switches under ED/LL and ED2/LL is closer to that of LLA as λ increases. When the system is highly overloaded, the tasks with zero laxity occur often so that the preemptions by the tasks is increased under ED/LL and ED2/LL.

5.4 Effects of the Laxity Values

The following parameters are used in this experiment: m=6, λ=0.4, and the standard deviation of the laxity value is 2. ***UB*** is 0.8, and the number of tasks generated is 200. Figure 7 shows the effects of laxity values on SR and CS. Laxity values are varied from 2 to 12. LLA, EDZL, ED/LL, and ED2/LL are valid for the given set of tasks when $L \geq 12$, while EDF and EDA2 can be so when $L \geq 14$. SR is not much affected by the value of laxity. From figure 5 and 6, we can see that SR is more susceptible to the number of processors and the arrival rate of tasks.

CS under LLA increases as L increases, while CS under the remaining five algorithms is relatively uniform. As the laxity value is large, it takes more time until the task is completed or the laxity value reaches zero. Therefore, more preemptions occur under LLA, and less preemptions under ED/LL and ED2/LL. However, if L becomes really large ($L \geq 11$), less preemptions occur even under LLA, because of reducing the competition between old tasks and new tasks.

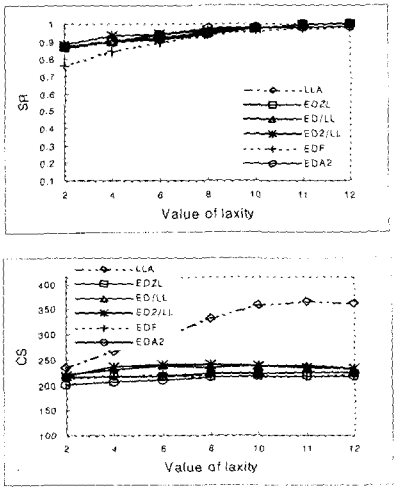

Fig 7 Effects of the laxity value (L)

From the experimental results, we can see that the proposed algorithms ED/LL and ED2/LL perform well in comparison with the existing algorithms. ED/LL and ED2/LL are suboptimal on m-multiprocessors because they can generate a valid schedule for every feasible set of ready tasks. ED/LL and ED2/LL show the same validity as LLA in meeting the deadlines of tasks, while the number of context switches is much smaller than that of LLA. Especially, ED2/LL shows good performance independent of system load. EDF is ineffective algorithm on multiprocessors. EDA2 shows poor SR in normal load phase even though it shows very good performance in highly overload phase. Considering the overhead of context switches, ED/LL and ED2/LL are more desirable than LLA in real-time systems.

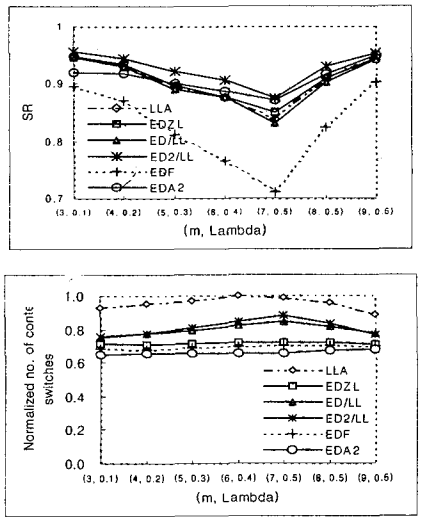

Fig 8 Effects of the fluctuant rate of task arrival

5.5 Effects of the load fluctuation

In this experiment, we checked the performance of each algorithm with changing workloads, arrival rates and the number of processors during runtime. If λ was used in this experiment, it means that we performed the experiment using λ for seventy percent of total tasks and (λ+0.3) for thirty percent. Than is, λ=0.2 means that the seventy percent of the tasks arrived at the rate of 0.2 and the thirty percent arrived at 0.5. We used the following parameters: the value of L = 4, and the standard deviation of laxity value is 1. The number of tasks generated is 1000.

Figure 8 shows the effects of fluctuant λ ($0.1 \leq \lambda \leq 0.5$) on SR and the normalized CS on m-processors ($3 \leq m \leq 9$). The *UB* is 0.8. As we can see from figure 8, ED2/LL can achieve the largest SR and show much smaller CS than those of LLA when the system is overloaded with fluctuant λ. Generally the system load is not fixed but is fluctuated depending on work time in real systems. Under this condition, EDA2 could not achieve good performance even in overload phase. LLA, EDZL, and ED/LL show better performance than EDA2 in lightly overload phase.

Table 5 Effects of utilization bound under ED2/LL

UB	ED2/LL (m=4, λ=0.2)		
	SR	CS	PE
1.0	0.930	1172	242
0.9	0.945	1176	217
0.8	0.945	1176	217
0.7	0.941	1150	189
0.6	0.941	1150	189
0.5	0.934	1084	126
0.4	0.934	1084	126
0.3	0.934	1084	126
0.2	0.918	989	31
0.1	0.918	989	31

We also analyzed the performance of ED2/LL by changing *UB* on 4-processor system when λ=0.2. The experimental results are shown in table 5. SR is not much affected by *UB*. The number of context switches also decreased as UB decreased. This is because ED2/LL get more chance to use EDA2 when *UB* is small. Considering SR, the overhead of context switches, and system load phase together, ED2/LL is much more desirable than LLA, EDA2, and EDZL in real-time systems.

6. Conclusions

In this paper, we have proposed new on-line algorithms ED/LL and ED2/LL for real-time multiprocessors. In multiprocessor systems, EDF has a low context switching overhead, but suffers from multi-processor anomalies in that it has a high

deadline miss ratio. LLA has been shown as suboptimal, but has a potential high context switching overheads. EDZL is suboptimal only for two processors and EDA2 is not suboptimal for multiprocessors. The ED/LL algorithm proposed in this paper has advantages from both EDF and LLA: low context switching overhead and low deadline miss rate. We also show that ED/LL is suboptimal for multiprocessors. Since ED/LL is not good when the system is overloaded, we also proposed ED2/LL which uses EDZL or ED/LL in normal load phase and uses EDA2 in overload phase. Finally, simulation results show that ED/LL and ED2/LL are practical algorithms that can be used in real-time multiprocessor systems.

References

[1] R. Clark, "Scheduling dependent real-time activities", PhD thesis, Computer Science Department, Carnegie-Mellon University, 1990.

[2] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, "On-line scheduling in the presence of overload", 1991 IEEE Symposium on Foundations of Computer Science, pp.100-110, Oct. 1991.

[3] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, "On the competitiveness of on-line real-time task Scheduling", Proc. of 12th Real-Time Systems Symp., pp.106-115, Dec. 1991.

[4] C. D. Locke, "Best-effort decision making for real-time scheduling", PhD thesis, Computer Science Department, Carnegie-Mellon University, 1986.

[5] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM, vol.20, no.1, pp.46-61, 1973.

[6] M. L. Dertouzos and A. K. Mok, "Multiprocessor on-line scheduling of hard-real-time tasks", IEEE Trans. on Software Engineering, vol.15, no.12, pp.1497-1506, Dec. 1989.

[7] J. Y.-T. Leung, "A new algorithm for scheduling periodic, real-time tasks", Algorithmica, vol.4, pp.209-219, 1989.

[8] A. Srinivasan and S. Baruah. "Deadline-based Scheduling of Periodic Task Systems on Multiprocessors", Information Processing Letters. Accepted for publication.

[9] S. K. Lee and D. Epley, "On-line scheduling algorithms of real-time sporadic tasks in multiprocessor systems", Technical Report 92-3, Computer Science, University of Iowa, 1992.

[10] S. K. Lee, "On-line multiprocessor scheduling algorithms for real-time tasks", Proc. IEEE Region 10's Ninth Annual International Conf., pp.607-611, Aug. 1994.

[11] S. Cho, S. K. Lee and H. Yoo, "On-line scheduling algorithms for hard real-time sporadic tasks", Journal of KISS(A), Vol. 25, No. 7, pp. 708-718, July 1998.

[12] H. Chetto and M. Chetto, "Some results of the earliest deadline scheduling algorithm", IEEE Trans. Software Engineering, vol.15, no.10, pp.1261-1269, Oct. 1989.

[13] K. Jeffay, "Scheduling sporadic tasks with shared resources in hard-real-time systems", Proc. of 13th IEEE Real-time Systems Symp., pp.89-99, Dec. 1992.

[14] S. Baruah, "Fairness in periodic real-time scheduling", Proc. of 16th IEEE Real-time Systems Symp., pp.200-209, Dec. 1995.

[15] S. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel, "Proportionate progress: a notion of fairness in resource allocation", Algorithmica, vol.15, pp.600-625, 1996.

[16] S. Baruah, J. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources", Proc. 9th International Parallel Processing Symp., pp.280-288, 1995.

[17] I. Ripoll, A. Crespo, and A. Garcia-Fornes, "An optimal algorithm for scheduling soft aperiodic tasks in dynamic-priority preemptive systems", IEEE Trans. on Software Engineering, vol.23, no.6, pp.388-400, Jun. 1997.

조 성 제

1989년 서울대학교 컴퓨터공학과 졸업. 1991년 서울대학교 컴퓨터공학과 석사학위 취득. 1996년 서울대학교 컴퓨터공학과 박사학위 취득. 1996년. 9월 ~ 1997년. 8월 서울대학교 컴퓨터신기술연구소 연구원. 2001년. 3월 ~ 2002년. 2월 미국 University of California, Irvine 객원 연구원. 1997년~현재 단국대학교 정보컴퓨터학부 조교수로 재직 중. 관심분야는 운영체제, 실시간 시스템, 컴퓨터 보안, 분산시스템 등