Journal of Universal Computer Science, vol. 17, no. 6 (2011), 912-925
submitted: 15/5/10, accepted: 30/11/10, appeared: 28/3/11 © J.UCS

Least Slack Time Rate First: an Efficient Scheduling Algorithm for Pervasive Computing Environment

Myunggwon Hwang
(Korea Institute of Science and Technology Information, Daejeon, South Korea
mgh@kisti.re.kr, mg.hwang@gmail.com)

Dongjin Choi
(Chosun University, Gwangju, South Korea
Dongjin.Choi84@gmail.com)

Pankoo Kim[1]
(Chosun University, Gwangju, South Korea
pkkim@chosun.ac.kr)

Abstract: Real-time systems like pervasive computing have to complete executing a task within the predetermined time while ensuring that the execution results are logically correct. Such systems require intelligent scheduling methods that can adequately promptly distribute the given tasks to a processor(s). In this paper, we propose LSTR (Least Slack Time Rate first), a new and simple scheduling algorithm, for a multi-processor environment, and demonstrate its efficient performance through various tests.

Keywords: Least Slack Time Rate First, multi-processor scheduling, pervasive computing environment, scheduling algorithm
Categories: C.1, C.2

1 Introduction

Recently, studies of pervasive computing are in progress for various purposes. Pervasive computing environment (PCE) aims at real-time interaction between human and computer by collecting and integrating all the information related to daily life, and provides intelligent services for the people's convenience and the prevention from dangerous situations. PCE is already involved deeply into many areas such as houses, hospitals (Jara, 2010), and schools, and into many devices like mobile devices (Costa, 2010), cars, medical assistances, and emergency detectors (Brooks, 1997, Mann, 2001, Su-Jin, 2002, Xiong, 2009). As having to complete processing the information inputted from various sensors and provide intelligent service before the user leaves from the terminal, pervasive computing requires more prompt and accurate data processing within predetermined deadline than general computers do, otherwise the result can even cause serious damage to human life. In this point, pervasive computing can be considered as a representative type of real-time system. Furthermore, as people's behaviour pattern consists of a wide range of information, it

[1] Corresponding author

has to promptly process user information including behaviour and position and continuously provide the required services. To implement various functions available for the user requires a processor scheduler that can provide services in a timely manner by effectively and intelligently processing the information inputted from the user. Moreover, there are numerous sensor data that pervasive computing tools need to process and sometimes the systems require more than one processor. However, the previously proposed multi-processor scheduling algorithms (Stavrinides, 2009, Stavrinides, 2010) require complicated operations and thus are rarely appropriate for pervasive computing environment which needs to be lightweight. Due to such characteristics of pervasive computing, in this paper a simple scheduling algorithm is proposed appropriate for multi-processor environment.

This paper is an expansion of the paper (Hwang, 2010) presented in the workshop (IMIS 2010) held as part of International Conference on CISIS 2010. In the previous paper, all tests have demonstrated the probability of 100% scheduling within the predetermined deadline. In the process of preparing this paper, however, it appeared that it was a rather hasty conclusion because the tests were performed for a small number of task sets and a wide range of utilization. Therefore, this paper revises part of LSTR described in the previous paper and evaluates the algorithm by applying it to more task sets and more specified utilization range.

This paper is organized as follows: Chapter 2 describes the fundamental studies of this research. In chapter 3, we indicate the limitations of existing algorithms and then propose LSTR scheduling algorithm. Further, we show the experimental results and performance evaluation in the fourth chapter. Finally, we summarize our research in the fifth chapter.

2 Fundamental Studies

This chapter describes general characteristics of tasks and the scheduling algorithms that served as a motive for LSTR algorithm.

2.1 Tasks and Schedulers

All tasks in real-time systems have timing constraints such as release time, relative deadline, absolute deadline, and execution time, as shown in Figure 1. The release time (r_i) represents the time when a task arrives at the ready queue for execution; the relative deadline (D_i, or d_i-r_i) is the maximum amount of time within which a task should be completed; the absolute deadline (d_i, or D_i+r_i) is the time within which execution of a task should be completed; and lastly, the execution time (e_i) represents the amount of time (theoretical minimum execution time) required for the entire task process.

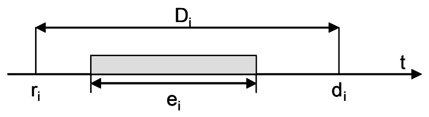

Figure 1: Timing constraints: release time, relative deadline, absolute deadline, and execution time. This figure is excerpted from (Hwang, 2010).

Many tasks occur simultaneously and each has its own timing constraints. For satisfying these constraints, especially task deadline, a system requires scheduling methods that allocate tasks to appropriate processor(s). According to the scheduling methods used, scheduling algorithms are classified into dynamic and static (fixed) priority scheduling algorithms. Dynamic priority scheduling algorithms such as EDF (Earliest Deadline First) and LST (Least Slack Time first) assign different priorities to an individual job in each task. On the other hand, static priority scheduling algorithms such as RM (Rate Monotonic) and DM (Deadline Monotonic) assign the same priority to all jobs in each task (Labrosse, 2002, Liu, 2000, and Stallings, 2004). In this paper, we deal with the dynamic priority algorithms only.

2.2 Earliest Deadline First (EDF)

EDF is a very simple and famous algorithm in which the earlier the deadline is, the higher the priority is and the scheduler operates when a job is completed on a processor or when a task wakes up in the ready queue. Let us assume three tasks with time constraints in a ready queue, as shown in Table 1. Each task is scheduled and divided into several jobs. According to the timing constraints listed in Table 1, the scheduling process is carried out as outlined in Table 2.

Tasks	r_i	D_i	e_i
T1	0	12	3
T2	0	6	3
T3	0	4	1

Table 1: Three tasks with timing constraints (r_i: release time, D_i: relative deadline, e_i: execution time, the tasks are periodic and the deadline coincides with the period)

t	0	1	2	3	4	5	6	7	8	9	10	11
T1	12	11			8	**7**	**6**		-		-	-
T2	6	**5**			-	-	6		**4**			-
T3	**4**	-		-	**4**	-	-		4			**1**

Table 2: Priority calculating process based on EDF algorithm (t: current time)

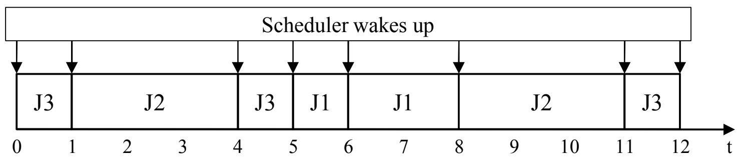

Figure 2: EDF scheduling result (J: a job of a task)

The three tasks are released at the initial time (t = 0). The scheduler attempts to compare the deadline of each task and then assigns the highest priority to T3 because its deadline is the earliest as value 4. Therefore, a job (J3) of T3 is executed on the processor. At t = 1, the scheduler wakes up since T3 has just completed. The scheduler then determines that T2 has the highest priority. At t = 4, T2 is completed and T3 is awakened due to its period, and between T1 and T3, the scheduler selects T3 (T2 is already finished at t = 4). T1 is then processed by the scheduler for the first time when t = 5. At t = 6, T2 is awakened due to its period and the priorities are the same between T1 and T2. In this case, the scheduler selects T1 because of tie-break rule[2]. At t = 8, the scheduler completes all the works of T1 and T2 is executed due to the tie-break rule again. After T2 is completed, T3 is carried out on the processor at t = 11. The three tasks are processed while each timing constraint is satisfied through iterative operations. Therefore, the EDF scheduling algorithm can be considered to be an optimal algorithm only for a single processor.

It is possible to determine whether given tasks are schedulable or not, by measuring processor utilization using timing constraints of given tasks without simulations. The processor utilization is measured by (1).

$$\Delta u = \sum_{i=1}^{n} \frac{e_i}{D_i} \leq 1 \tag{1}$$

Equation (1) measures processor utilization, where □u denotes the utilization; D_i the relative deadline; e_i the execution time of each task; and n the total count of tasks. If the utilization of given tasks is 1, the processor should run without idle state for satisfying each deadline. The utilization of the tasks listed in Table 1 is 1 (3/12 + 3/6 + 1/4 = 1). Therefore, we can assume that these tasks are schedulable.

2.3 Least Slack Time First (LST)

LST algorithm follows a rule that the smaller slack time, the higher priority. The slack time means the remaining spare time (d_i - e_i^r - t) at the current time. The e_i^r represents the time required to complete the remaining work of a task as shown in Figure 2. LST algorithm can schedule all tasks under the condition of satisfying (1). And, it can be considered to be the optimal algorithm for a single processor.

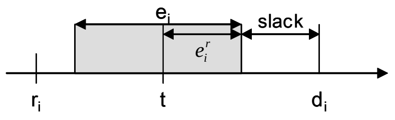

Figure 3: Slack is d_i - e_i^r - t. This figure is excerpted from (Hwang, 2010).

3 Scheduling Policies

In this chapter, we analyze the existing scheduling algorithms described in the previous chapter and examine their limitations in a multi-processor environment. We then propose an efficient scheduling algorithm that has the optimal possibility within

[2] Tie-break rule: the scheduler selects a task having the first index or the processed jobs at the last processing time.

a limited range of utilization for real-time systems like pervasive computing. Figure 4 describes the scheduling operation in pervasive computing.

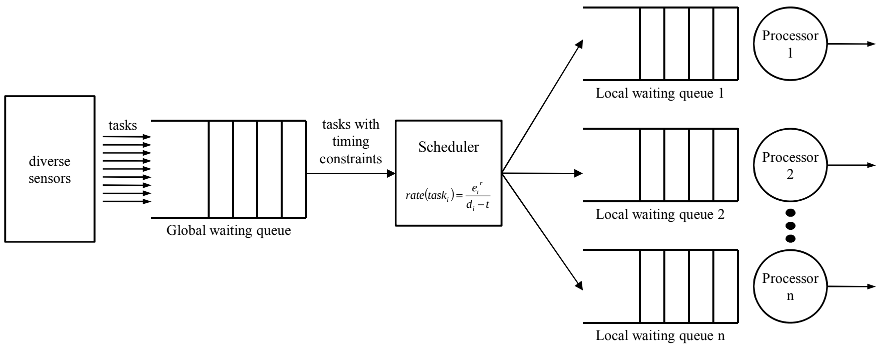

Figure 4: Scheduler model for pervasive computing environment

3.1 Limitations of existing algorithms

Algorithms described in the previous chapter are optimal in the uni-processor environment but not in the multi-processor one. Table 4 lists the results of scheduling on two processors using EDF according to timing constraints listed in Table 3. For the optimal scheduling, the utilization of each processor should be 1 but processor 2 is idle at times (*t*) 5 and (t) 11. In case LST is used, it also leaves some processors idle. Therefore, missing deadlines occur.

Tasks	r_i	D_i	e_i
T1	0	2	1
T2	0	3	2
T3	0	12	10

Table 3: Three tasks with timing constraints

t		0	1	2	3	4	5	6	7	8	9	10	11
T	**T1**	**2**	-	**2**	-	**2**	-	**2**	-	**2**	-	**2**	-
	T2	**3**	**2**	-	**3**	**2**	-	**3**	**2**	-	**3**	**2**	-
	T3	12	**11**	**10**	**9**	8	**7**	6	**5**	**4**	**3**	2	**1**
P	**P1**	J1	J2	J1	J2	J1	J3	J1	J2	J1	J2	J1	J3
	P2	J2	J3	J3	J3	J2	*IS*	J2	J3	J3	J3	J2	*IS*

Table 4: Scheduling process and result on two processors based on EDF (P: processor IS: idle state). If missing deadline occurs, the task is discarded for periodic task at its relative deadline.

Missing deadlines frequently occur because the scheduling algorithms depend on release time or slack time of given tasks. Further, scheduling based on these factors cannot exactly determine which task should be carried out first. Upon this, we suggest a new scheduling algorithm that can assign all of jobs on processor(s) while minimizing idle state.

3.2 Least Slack Time Rate first (LSTR): Prerequisites

LSTR scheduling algorithm has the same prerequisites as those for the existing scheduling algorithms for real-time systems. These prerequisites are listed in Table 5.

Conditions	Prerequisites
Pre-emption	Accept
Migration	Accept
Periodic task	Only
Release time	0
Timing constraints	Task set should satisfy (2)
Scheduling time	Basic time unit

Table 5: Prerequisites for LSTR scheduling algorithm

The first condition is pre-emptive scheduling, which means that the execution of jobs can be interleaved. The scheduler suspends the execution of less urgent jobs and gives processor control to a more urgent job. Later, after the urgent job is executed, the scheduler returns processor control to the previous job. The second prerequisite is migration. In the multi-processor environment, jobs that are ready for execution are placed in the priority queue. When a processor is available, the job at the head of the queue is carried out on the processor. In other words, an available processor can immediately execute any job in the ready queue. The third and fourth prerequisites, we assume that all tasks are periodic and the release time is 0 for well-understood behaviour of the algorithm. The fifth prerequisite is that all timing constraints have to satisfy (2).

$$\Delta u = \sum_{i=1}^{n} \frac{e_i}{D_i} \le n_c, \qquad \frac{e_i}{D_i} \le 1 \tag{2}$$

where, n_c denotes the number of processors. Equation (2) represents the ranges of timing constraints. The total processor utilization of a given task set cannot be higher than the number of processors and the processor utilization of each task is not more than 1. If the utilization of a task equals to 1, it implies that the task should occupy one processor till completion. In addition, if the total utilization equals to the number of processors, optimal scheduling is possible only if all processors work without idle state. And the last prerequisite is scheduling time which awakens the scheduler. Every scheduling algorithm has scheduling time that is used by the scheduler to determine which task should be executed first. Generally, the scheduler operates when tasks are released, when the execution of a task is completed, and when a task is awakened in the ready queue (when the time for its execution has arrived). However, LSTR scheduling algorithm operates on every basic time unit. Here, we consider the basic time unit of 1 (ms).

The six prerequisites discussed above are fundamental for dynamic scheduling algorithm in real-time systems, and the scheduling algorithm described in this paper also depends on these prerequisites.

3.3 Least Slack Time Rate first (LSTR): Scheduler

LSTR scheduling algorithm measures the rate of execution time of each task at every scheduling time. The aim of LSTR is to minimize that any processor has idle state. All tasks have deadline and execution time as timing constraints. The scheduler determines at the scheduling time which task to be executed on a processor. Tasks are carried out on the processor(s) and both the remaining execution time and the remaining deadline of these tasks decrease accordingly. However, the remaining execution times of other tasks do not decrease. This explains the limitations of existing scheduling algorithms such as EDF and LST which use only the deadline, release time, or slack time. Therefore, these algorithms cannot determine which task is really urgent. Also, in these algorithms, once the scheduler determines a task with the highest priority, others cannot be carried out for some fixed time during which the task is executed. This causes some processors to be in idle state. On the other hand, the LSTR scheduling algorithm determines the priorities of tasks by computing the rates between their remaining execution times and remaining deadlines so as to satisfy all of the timing constraints of the tasks. Equation (3) presents this scheduler. A task with a higher rate has a higher priority.

$$rate(task_i) = \frac{e_i^r}{d_i - t} \tag{3}$$

Although this is a simple equation, it can schedule tasks while minimizing idle states of processors. Here, t denotes the current time; e_i^r the remaining execution time of i-th task at t; and d_i the remaining absolute deadline of i-th task at t. The LSTR algorithm is similar to the LST algorithm which uses the slack time, but different in that it uses the rate of remaining execution time and remaining deadline. This is why we named this algorithm Least Slack Time Rate first. As described in the previous chapter, the scheduler operates at every basic time unit, which is 1. In our tests under these conditions, LSTR returned almost all of optimal scheduling results in both uni- and multi-processor environments. As an example, Table 6 shows the scheduling results using the timing constraints of Table 3 (three tasks) for multi-processor environment (two processors). As shown in the Tables 6, no processor has an idle state.

t		0	1	2	3	4	5	6	7	8	9	10	11
T	**T1**	1/2	**1/1**	1/2	**1/1**	1/2	**1/1**	1/2	**1/1**	1/2	**1/1**	**1/2**	-
	T2	**2/3**	1/2	**1/1**	2/3	**2/2**	**1/1**	**2/3**	1/2	**1/1**	**2/3**	1/2	**1/1**
	T3	**10/12**	**9/11**	**8/10**	**7/9**	**6/8**	6/7	**5/6**	**4/5**	**3/4**	2/3	**2/2**	**1/1**
P	**P1**	J3	J1	J2	J1	J2	J1	J3	J1	J2	J1	J3	J2
	P2	J2	J3	J3	J3	J3	J2	J2	J3	J3	J2	J1	J3

Table 6: LSTR based scheduling process and result on two processors

4 Experiment and Performance Evaluation

In the previous chapter, we presented the successful scheduling results of the given tasks using LSTR algorithm. In order to evaluate this algorithm, we implemented a simulation system and tested it using total 3,310,000 task sets. Further, in order to measure the amount of elapsed time by the LSTR algorithm, we evaluated its performance by comparing it with the EDF algorithm. For the purpose of test, we used a computer system with 4GB memory and two 2.80GHz CPU running Vista OS. Even though the system has two processors, the simulation was carried out on one processor.

4.1 Scheduling Results

In order to evaluate the scheduling performance of LSTR algorithm, we tested the algorithm for the uni-processor and the multi-processor environments. We tried to apply various conditions according to the numbers of processes and tasks in each environment, which are summarized in Table 7.

Num. of processors (*p*)	**Num. of tasks**													
	2	**3**	**4**	**5**	**7**	**9**	**11**	**12**	**13**	**15**	**17**	**20**	**23**	**25**
1 (x10000)	7	7	7	7	7	7	-	-	-	-	-	-	-	-
2 (x10000)	-	9	9	9	9	9	9	-	5	3	-	-	-	-
3 (x10000)	-	-	9	9	9	9	9	-	9	3	3	-	-	-
4 (x10000)	-	-	-	9	9	9	9	-	9	9	3	3	-	-
5 (x10000)	-	-	-	-	9	9	9	-	9	9	9	3	3	-
7 (x10000)	-	-	-	-	-	9	-	9	-	9	-	9	-	9

Table 7: Pairs of processor(s) and tasks (The number in each rectangle for the pair (p, t) means total count of task sets used for the scheduling test, where p and t are numbers of processor(s) and task sets respectively. For example, it is 7 in the rectangle (1, 5) and means that the simulation system uses 70,000 different task sets which consist of 5 tasks.)

As shown in Table 7, the scheduling performance was tested with the varying number of processors and tasks. For 1, 2, 3, 4, 5, and 7 processors, 420,000, 620,000, 600,000, 600,000, 600,000, and 450,000 task sets were used respectively. In addition, a random number creator was implemented and it could create timing constraints for processor-task pairs listed in Table 7 according to the utilization ranges. The utilization range of each processor environment and its scheduling result are described in respective sections.

4.1.1 Results for uni-processor environment

To test the scheduling performance for the uni-processor environment, a total of 420,000 task sets were created, which were grouped into 10,000 task sets respectively according to the utilization range. Table 8 shows the composition of task sets to be scheduled in the uni-processor environment.

Num. of tasks (*t*)	**△ *u* (utilization) ranges, *p = 1***						
	0.5< △ *u* ≤0.6	**0.6< △ *u* ≤0.7**	**0.7< △ *u* ≤0.8**	**0.8< △ *u* ≤0.9**	**0.9< △ *u* ≤0.95**	**0.95< △ *u* ≤0.98**	**0.98< △ *u* ≤1.0**
2	10,000	10,000	10,000	10,000	10,000	10,000	10,000
3	10,000	10,000	10,000	10,000	10,000	10,000	10,000
4	10,000	10,000	10,000	10,000	10,000	10,000	10,000
5	10,000	10,000	10,000	10,000	10,000	10,000	10,000
7	10,000	10,000	10,000	10,000	10,000	10,000	10,000
9	10,000	10,000	10,000	10,000	10,000	10,000	10,000

Table 8: Total counts of task sets given according to number of tasks and utilization ranges. The sum of task sets in the same row is exactly same to the total count of tests for pair (p, t) in table 7 (in here, p is 1).

LSTR scheduler made the optimal scheduling result for all the task sets given above, which confirms that LSTR scheduler could make the optimal result in the uni-processor environment under the condition where the utilization is 1 or less, like EDF scheduler.

4.1.2 Results for multi-processor environment

Though being capable of making the optimal scheduling result in the uni-processor environment, LSTR algorithm was originally designed for high-performance scheduling results in the multi-processor environment. In order to test its performance under the multi-processor environment, the LSTR algorithm was tested for 10,000 task sets for each condition consisting of tasks, processors and utilization ranges. The utilization ranges used for the test are listed in Table 9.

△ *u* ranges	0.5< △ *u* ≤0.6	0.6< △ *u* ≤0.7	0.7< △ *u* ≤0.8	0.8< △ *u* ≤0.9	0.9< △ *u* ≤0.95	0.95< △ *u* ≤0.98	0.98< △ *u* ≤0.99	0.99< △ *u* ≤0.995	0.995< △ *u* ≤1.0

Table 9: utilization ranges for each pair (p, t), where p is 2 or more. And, according to utilization range and pair (p, t), 10,000 task sets are prepared.

For all task sets whose utilization ranges were 0.99 or lower, the LSTR algorithm appeared to meet the predetermined deadline. For those whose utilization ranges were higher than 0.99, however, it failed to meet deadlines in very small parts. Table 10 summarizes the parts where the algorithm missed the deadline.

⊿ *u* ranges		**0.99< ⊿ *u* ≤ 0.995**					**0.995< ⊿ *u* ≤ 1.0**				
Num. of Processors		**2**	**3**	**4**	**5**	**7**	**2**	**3**	**4**	**5**	**7**
Num. of Tasks	3	0	-	-	-	-	0	-	-	-	-
	4	0	0	-	-	-	0	0	-	-	-
	5	0	0	0	-	-	0	0	0	-	-
	7	0	0	0	0	-	0	0	0	0	-
	9	0	0	0	0	0	0	0	**1**	**1**	0
	11	0	**1**	0	0	-	**4**	0	**2**	0	-
	12	-	-	-	-	0	-	-	-	-	0
	13	0	0	0	0	-	0	0	**1**	**1**	-
	15	0	0	0	0	0	0	0	**15**	0	0
	17	-	0	0	0	-	-	0	0	**1**	-
	20	-	-	0	0	0	-	-	0	0	0
	23	-	-	-	0	-	-	-	-	**3**	-
	25	-	-	-	-	0	-	-	-	-	0

Table 10: Failure cases: in case of pair (4, 15) in 0.995< ⊿ u≤ 1.0, the number in the rectangle is 15. It means that its failure rate is 0.15(%) because 10,000 different task sets were used for each test.

However, the percentage of the LSTR algorithm not meeting the deadline was extremely low and it produced the optimal results when the utilization ranges were 0.99 or lower. Considering that 10,000 task sets were used for each pair (p, t) and utilization range, it can be said that such result proves high reliability of the LSTR algorithm. Table 11 shows some of task sets of which deadlines were missed.

⊿ *u* ranges	**Pair(p,t)**	**Task sets (D, e)**					
0.99~0.995	Pair(3,11)	(4,1)	(4,1)	(12,3)	(3,1)	(3,1)	(3,1)
	⊿ ***u=0.994***	(3,1)	(20,4)	(4,1)	(20,4)	(12,3)	
0.995~1.0	Pair(2,11)	(32,4)	(6,1)	(40,5)	(4,1)	(6,1)	(16,4)
	⊿ ***u=1.000***	(4,1)	(12,2)	(6,1)	(6,1)	(12,2)	
	Pair(4,9)	(20,6)	(8,3)	(16,5)	(4,2)	(8,4)	(8,4)
	⊿ ***u=0.997***	(2,1)	(2,1)	(2,1)			
		(32,4)	(10,2)	(24,3)	(4,1)	(10,2)	(5,1)
	Pair(5,23)	(8,2)	(8,2)	(10,2)	(10,2)	(4,1)	(8,2)
	⊿ ***u=1.000***	(4,1)	(4,1)	(8,2)	(4,1)	(10,2)	(5,1)
		(5,1)	(5,1)	(8,2)	(4,1)	(5,1)	

Table 11: Task sets missing deadline

4.2 Elapsed Time Evaluation

In the previous section, we demonstrated the scheduling performance of LSTR algorithm. However, this algorithm would have scheduling time more frequent than those of other algorithms such as EDF and LST. Schedulers of other scheduling algorithms wake up when a job on a processor is released or completed, and when a task of higher priority wakes up. In case of LSTR, however, the scheduler wakes up at every basic time unit. Therefore, we need to know how much time is wasted by the LSTR. We have attempted to evaluate the elapsed time by comparing LSTR with EDF because this is one of the simplest and the fastest. However, it is not optimal algorithm for the multi-processor environment. So, we evaluated the difference of elapsed time under a single-processor environment using task sets which consists of three, five, seven, and nine tasks respectively, and with the utilization range of 0.95 or higher.

In the computer system, a measurable time unit is 1 ms (0.001 second); however, the scheduling time for one task set is smaller than 1 ms. Therefore, we needed to iterate each task set 10,000 times to obtain appropriate values. We prepared 200 task sets consisting of 50 sets of three, five, seven, and nine tasks respectively. Using these sets, both the EDF and the LSTR simulators were operated under the same conditions.

The results of the performance evaluation are shown in Figure 5, where the x-axis represents the index of a task set and the y-axis the time required (ms) for scheduling. The scheduling time of LSTR is slightly higher than that of EDF, as indicated in Table 12 which presents the summarization of this performance evaluation. For the task sets consisting of three tasks, EDF spent 186 (ms) on average for scheduling; LSTR 281 (ms), showing the time difference of 95. As this was the result of iterating one task set 10,000 times, the actual elapsed time was 0.0186 (ms) and 0.0281 (ms) respectively, making the time difference 0.0095. Moreover, in case of the nine-task set which LSTR wasted the longest average time, its time difference with EDF is 0.0449. Considering that the time EDF spent for scheduling this task set was 0.197, the time difference is not weak point because LSTR has excellent performance under the multi-processor environment. It showed the 13th set of 9 tasks (EDF: 2061 (ms) and LSTR: 3731 (ms)) had the greatest time difference of 1670 (ms). For this set, the actual time difference was 0.167 (ms) and LSTR spent more time. These results indicate that LSTR wastes slightly more scheduling time than EDF does. Still, LSTR is more valuable because it shows the possibility of optimal scheduling with the utilization range of 0.99 or less in the multi-processor environment.

Num. of tasks	**3 tasks**		**5 tasks**		**7 tasks**		**9 tasks**	
Method	**EDF**	**LSTR**	**EDF**	**LSTR**	**EDF**	**LSTR**	**EDF**	**LSTR**
Avg. ET	186	281	518	762	1,074	1,436	1,970	2,419
Worst ET	761	910	1,685	2,321	2,886	3,160	6,928	7,571
Difference (LSTR-EDF)	95 (0.0095 ms)		254 (0.0254 ms)		363 (0.0363 ms)		449 (0.0449 ms)	

Table 12: Scheduling process using EDF algorithm (ET: Elapsed Time)

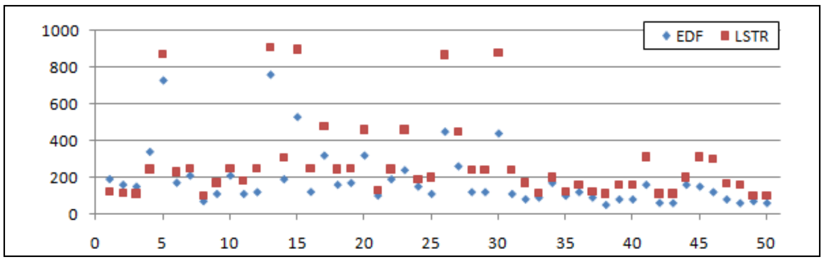

Figure 5-1. 50 sets of 3 tasks

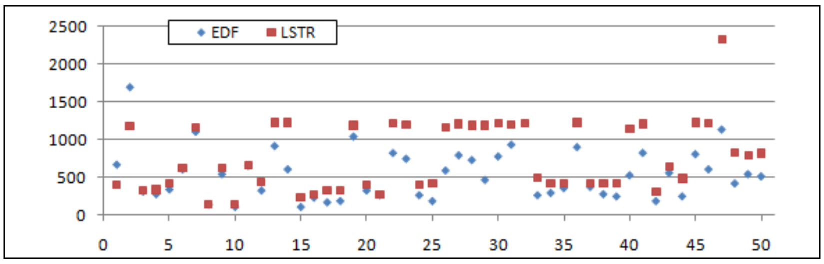

Figure 5-2. 50 sets of 5 tasks

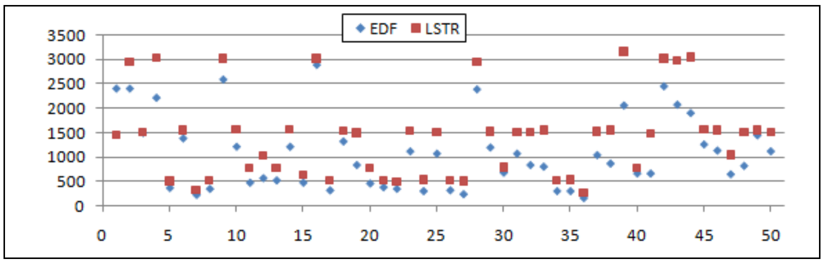

Figure 5-3. 50 sets of 7 tasks

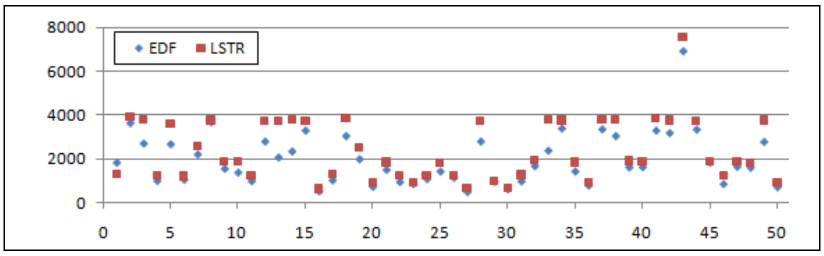

Figure 5-4. 50 sets of 9 tasks

Figure 5: Comparison results for elapsed time, x-axis: index of task set, y-axis: elapsed time (ms)

5 Conclusions

This paper proposed LSTR scheduling algorithm which has excellent performance in the multi-processor environment. The LSTR algorithm overcomes the limitations that EDF and LST which are optimal schedulers in the uni-processor environment have in the multi environment, as well as satisfying the pre-determined deadlines of all tasks in the uni-processor environment. In the multi processor environment, it produced the optimal scheduling result for the utilization range of 0.99 or less and at least 99.85(%) success rate at the utilization range higher than 0.99. It appears that LSTR spends more scheduling time than the existing scheduling algorithms do because it measures the priorities of tasks at every basic time unit. However, it is proven through the experiments that LSTR has slightly higher elapsed time than EDF algorithm does. In other words, though LSTR spends a little bit more scheduling time, it still can be considered as a better scheduling algorithm as it has higher performance in the multi-processor environment.
Even though it is expected that LSTR scheduler can be utilized in diverse field for real-time services like pervasive computing as well as for data transmission in general processors and networks, it is still necessary to further study the resource management for job migration and theoretic proof. We will continuously research on it.

Acknowledgements

This research was financially supported by the Ministry of Education, Science Technology (MEST) and National Research Foundation of Korea (NRF) through the Human Resource Training Project for Regional Innovation.

References

[Brooks, 1997] Brooks, R.A.: The Intelligent Room Project, Second International Conference on Cognitive Technology 1997: 'Humanizing the Information Age,' pp. 271~278, 1997.

[Costa, 2010] Costa, G., Lazouski, A., Martinelli, F., Matteucci, I., Issarny, V., Saadi, R., Dragoni, N., and Massacci, F.: Security-by-Contract-with-Trust for Mobile Devices, Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, Vol. 1, No. 4, pp. 75~91, 2010.

[Hwang, 2010] Hwang, M.G., Choi, D.J., and Kim, P.K.: Least Slack Time Rate first: New Scheduling Algorithm for Multi-Processor Environment, CISIS 2010, pp. 806~811, Feb 2010.

[Jara, 2010] Jara, A. J., Zamora, M. A., and Skarmeta, A. F. G.: An Initial Approach to Support Mobility in Hospital Wireless Sensor Networks based on 6LoWPAN (HWSN6), Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, Vol. 1, No. 2/3, pp. 107~122, 2010.

[Labrosse, 2002] Labrosse, J.J, MicroC OS-II: The Real-Time Kernel (Second Edition), CMPBOOKS, Manhasset NY, 2002.

[Liu, 2000] Liu, J.W.S., Real-Time System, Printice Hall, New Jersey, 2000.

[Liu, 2003] Liu, J. and Lee, E.A.: Timed Multitasking for Real-Time Embedded Software, IEEE Control System Magazine, pp. 65~75, Feb 2003.

[Mann, 2001] Mann, S., Wearable Computing: Toward Humanistic Intelligence, IEEE Intelligent Systems, 16(3), pp.10~15, 2001.

[Stallings, 2004] Stallings, W., Operating Systems: Internals and Design Principles (fifth international edition), Printice Hall, New Jersey, 2004.

[Stavrinides, 2009] Stavrinides, G.L. and Karatza, H.D., Fault-tolerant Gang Scheduling in Distributed Real-time Systems Utilizing Imprecise Computations, Simulation, 85(8), pp. 525~536, 2009.

[Stavrinides, 2010] Stavrinides, G.L. and Karatza, H.D., Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations, The Journal of Systems and Software, online published first, 2010.

[Su-Jin, 2002] Su-Jin, P.P., Lebeltel, O., and Laugier, C., Parking a Car using Bayesian Programming, ICARCV 2002, Vol. 2, pp. 728~733, 2002.

[Xiong, 2009] Xiong, N., He, J., Park, J.H., Cooley, D., and Li, Y., A Neural Network based Vehicle Classification System for Pervasive Smart Road Security, Journal of Universal Computer Science, 15(5), pp. 1119-1142, 2009.