A Heuristic Scheduling Algorithm for Reducing the Total Error of an Imprecise Multiprocessor System with 0/1 Constraint

Ki-Hyun Song, Kyung-Hee Choi, Seung-Kyu Park, Dug-Kyoo Choi, and Kyong-Ok Yun

Abstract

The scheduling problem of satisfying both 0/1 constraint and the timing constraint while minimizing the total error is NP-complete when the optional parts have arbitrary processing times. In this paper, we present a heuristic scheduling algorithm for 0/1 constraint imprecise systems which consist of communicating tasks running on multiple processors. The algorithm is based on the program graph which is similar to the one presented in[4]. To check the schedulability, we apply Lawler and Moore's theorem. To analyze the performance of the proposed algorithm, intensive simulation is done. The results of the simulation shows that the longest processing first selection strategy outperforms random or minimal laxity policies.

I. Introduction

The imprecise system, proposed in[2, 7], provides flexibility in scheduling time-critical tasks. Examples of its applications include image processing and tracking. Recently, the imprecise techniques are widely applied to the multimedia processing to provide the mechanism for QoS[1]. Several scheduling algorithms have been developed in this area. Those are the algorithms for scheduling imprecise jobs on uniprocessors, jobs with multiple versions, and those for scheduling multitasking computations on multiprocessor systems.

For some applications, execution of the optional parts is of value only if they are executed completely before the deadline, and of no value if they are executed partially. The systems with such imprecise tasks are called systems with 0/1 constraints. Most scheduling problems of satisfying both 0/1 constraint and timing constraints, while the total error is minimized, is NP-complete when the optional tasks have arbitrary processing times[7]. By the total error, we mean the sum of the processing times of all optional tasks that could not be scheduled. In[2], Liu suggested a reasonable strategy of scheduling tasks with the 0/1 constraint on uniprocessors for minimizing the total error. This method schedules the first optional task with the longest processing time. In the case of multiprocessor systems, one should take the communications and states of processors into consideration for the correct scheduling of optional tasks.

Natale proposed an end-to-end scheduling algorithm for distributed systems in which the communications are considered [4]. This algorithm shows good performance for precise systems in which the tasks do not include optional parts. Natale's algorithm, however, cannot be applied to the imprecise system since it does not provide a mechanism of reducing the program graph or time graph for the optional tasks[6]. A special mechanism is additionally required when the optional tasks are considered. The different criteria of schedulability should be also adopted. The scheduling problem in the precise systems is to determine whether it can schedule all the tasks or not. In the imprecise system on the other hand, several criteria can be used such as minimization of total error, minimization of the maximum or average error, etc.

In this paper, we present a heuristic scheduling algorithm for imprecise multiprocessor systems with 0/1 constraint which consists of a set of communicating tasks with 0/1 constraints. Each task has mandatory and optional tasks. They run on multiple processors. The tasks are statically assigned to given processors. As those in the previous works[4, 6], the proposed heuristic algorithms are based on the dual graphs, transformed from reduced program graphs. To obtain the reduced program graph, we apply a reducing algorithm which is a little modified from that in[4, 6].

The following shows steps of the proposed heuristic scheduling. First, we calculate modified ready times and modified deadlines of all tasks. Then an optional task is selected according to one of

Manuscript received January 30, 1997; accepted September 9, 1997.

K. H. Song, K. H. Choi, S. K. Park, and D. K. Choi were with School of Information & Computer Engineering, Ajou University, Suwon, Korea.

K. O. Yun is with ETRI, Taejon, Korea.

three different policies: the longest processing time first, the minimum laxity policy, or the random policy. Once an optional task is selected, whether it can be scheduled or not is checked based on its execution time, modified ready time and modified deadline. By the Lawler and Moore's theorem, tasks with modified ready times and deadlines can be scheduled if and only if tasks with the given ready times and deadlines can be also scheduled[3]. If it turns out that it cannot be scheduled with the selected option, the optional part is rejected. Otherwise, when an optional task is accepted, modified ready times and modified deadlines are re-calculated. We repeat these steps until all the optional tasks are checked.

To analyze the proposed algorithm, intensive simulations are done to see the total errors from the scheduling algorithm. The simulation model is the same as that of[4]. The processing times are randomly generated for the mandatory and optional parts of tasks. The simulation shows that the longest processing first policy outperforms the other two policies. When the number of tasks is small, no notable difference is observed. As the number of tasks increases, the total errors produced by the longest processing first policy shows the efficiency of the algorithm in which the amount of errors increases more slowly than the other policies. The scheduling with the longest processing first policy outperforms than the others, but is still not optimal in a point of view of the optimality.

The rest of this paper is organized as follows. In section II, we describe briefly the model of imprecise multiprocessor systems with 0/1 constraints introducing some notations. The proposed scheduling algorithm is presented in section III. The results of simulation and analysis are described in section IV. Section V concludes this paper.

II. System Model

The 0/1 constraint imprecise multiprocessor system is modeled in this paper as a directed graph G which is called the program graph. This is a dual directed graph as introduced in[4]. Each edge E_i (can appear more than once in dual graph G), a basic unit of computation or task, is characterized by the following parameters. (We will use the term, computation or task, interchangeably.)

- $P_j = \tau(E_i)$: means a processor P_j on which the computation E_i runs,
- $c_i(E_i)$: processing times that may be either m_i or $m_i + o_i$, where m_i and o_i are the processing time of mandatory part M_i and optional part O_i of E_i, respectively. m_i and o_i may take an arbitrary value.

Figure 1 shows one of possible models with 6 tasks, E_1 through E_6, running on two processors, P_1 and P_2. For instance, tasks 3, 4, 6 run on processor P_1, and task 1,2, and 5 run on processor P_2. Each node represented by N_k is used to connect edges. If E_i is an incoming edge to N_k ,and E_j is an outgoing

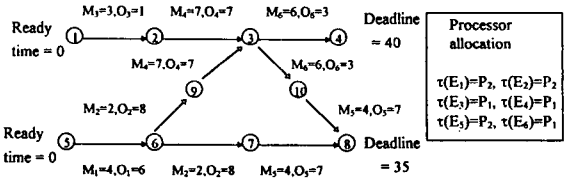


Fig. 1. A program graph G of system with 6 tasks.

edge from N_k, then computation E_i must be completed before the computation E_j starts its execution. In this case, the edge E_i is called an immediate predecessor of E_j , and the edge E_j is called an immediate successor of E_i. The precedence relations can be derived from the control of flows of the program graph or the communications. The sub-graph $G(P_i)$ is defined to be a subset of the graph G in such a way that (E_k) is equal to P_i for all E_k in $G(P_i)$, This is a sub-graph corresponding to a set of tasks that are running on processor P_i. Let G_m be a sub-graph of the program graph G where all the optional parts are ignored. It means $c_i(E_i) = m_i$ for all E_i in G_m. Let G_1 be a sub-graph of a program graph. Then, $G_2 = G_1 \cup \{O_i\}$ means that we add an optional part O_i to the edge E_i in G_1, which gives a sub-graph of G_2 where $c_i(E_i) = m_i + o_i$. In this case, we note $O_i \in G_2$

Let $Pred(E_i)$ be the set of all predecessor edges of E_i and let $Succ(E_i)$ be the set of all successor edges E_i. In our model, ready times are assigned to all nodes N_k that do not have any incoming edge. Deadlines are assigned to all nodes N_k that do not have any outgoing edge. The nodes that do not have incoming edges are called the starting nodes of the system while the nodes that do not have outgoing edges are called the ending nodes of the system. In the figure 1, for example, node 1 and 5 are starting nodes and nodes 4 and 8 are ending nodes. Initially, only these nodes have ready times or deadlines as shown in figure 1. The modified ready time r_i of each edge E_i is the maximum value among $d_k + m_k$ for all immediate predecessors E_k. The modified deadline d_i of a edge E_i is the minimum among $r_k - m_i$ for all immediate successors E_j.

A sub-graph is schedulable if all edges finish computations before their deadlines after immediate predecessors complete executions. If G_m is schedulable, we say that the imprecise system is minimally schedulable. If a sub-graph G' is schedulable, the total error of G' is defined as the sum of all o_i where $O_i \notin G'$. The optimal error of a graph G is defined as the total error of a sub-graph G' of G, where there is no sub-graph G'' of G such that G' is a sub-graph of G'' and G'' is also schedulable.

III. Scheduling Algorithm

The scheduling problem in this paper is a process to find a schedulable sub-graph of a program graph G. In general, the program graph can be constructed from source code with additional information like SDL[6]. Since the original program graphs

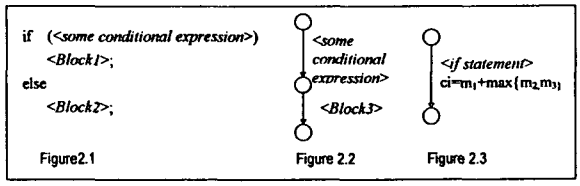


Fig. 2. Sample program and corresponding program graphs.

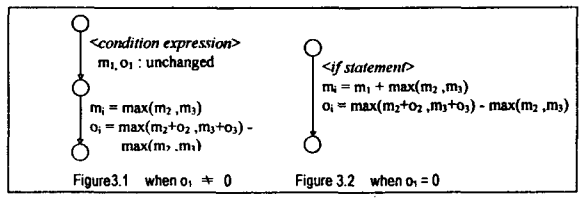


Fig. 3. Program graphs when tasks have optional parts.

in general are too complex to manipulate, graph reduction is required. One possible approaches is the method introduced by[6]. However, some modification is required to apply it to our algorithm. We will describe briefly the modification which explains how to reduce the original program graph. And then, our model of imprecise systems is derived.

When a task have mandatory and optional parts, we must consider the context in which the task is located. For instance, consider the classical program code in Figure 2.1. If <Block1> and <Block2> are precise blocks, (i.e, if they do not have any optional parts,) we can reduce them into one <Block3>, as shown in figure 2.2, whose computational requirement is equal to the maximum value among the computation requirement of <Block1> and that of <Block2>. Eventually the three computation units can be merged into one computation unit as depicted in Figure 2.3.

If <Block1> or <Block2> has optional parts, we cannot always reduce them into one block. Such blocks can be reduced into one of the program graphs as shown in figure 3. In figure 3, m_1, o_1, m_2, o_2, m_3, and o_3 are the computational requirements of the mandatory part and optional part of <some conditional expression>, <Block1>, and <Block2>, respectively.

Once the original program graph is reduced and translated to program graph G as the form of figure 3, we begin to find a schedulable sub-graph of G. Since the computational requirement of the optional part can be arbitrary in our model, the general problem of finding a sub-graph with the minimal error which is scheduled to minimize the total error is NP-complete[7]. So the heuristic algorithm might be a good candidate provided it produce a schedulable sub-graph whose total error is near or equal to the optimal total error. The proposed heuristic scheduling algorithm is as follows.

Suppose a program graph G, and a list L of optional parts are given. First, the modified ready times and modified deadlines for all nodes in G_m are calculated. Eventually the modified ready

```
Let G' = Gm, and L' =L ;
Calculate initial modified ready time and modified deadline for all nodes and edges;
while (L' ≠∅) {
  Select one Oi from L' according to the longest optional first strategy,
                                              and L' ← L' - {Oi};
  if (ri+mi+oi > di )
     Reject Oi and total error = total error + oi;
  else {
       Check schedulability of all tasks Ek in G' that are running on processor
       Pj = r (Ek) using theorem by Lawler and Moore, assuming ck(Ek) = mk + ok;
     if (schedulable) {
       G' = G' ∪ { Oi };
       ci (G') = mi + oi ;
       Let PS = { Ek | Ek ∈ Pred(Ei) ∪ Succ(Ei),
                                          where Ei ∈ PS};
         Update modified ready time and modified deadline for nodes from
           which edge in PS is going out or to which edge in PS is coming ;
     }
     else
       Reject Oi and total error = total error + oi;
  }
}
```

Fig. 4. Heuristic scheduling algorithm with the longest optional first strategy.

times and modified deadlines for all edges in G_m are also calculated. This calculation can easily be done. Next, we select an optional part O_i. The selection of O_i can be done by several ways. One is to select the longest one among the optional parts. Another way is to select randomly. Yet a third one is to select the one with minimum laxity. The path with minimum laxity here means a path where the difference, (the deadline of the ending node-the sum of the ready time of the starting node and the sum of the m_i in the path), yields minimum.

If the condition $r_i + m_i + o_i < d_i$ is not satisfied, then we can conclude that the optional part O_i cannot be scheduled in any way. Even though this condition is satisfied, one cannot conclude that O_i is schedulable. So we have to check whether O_i does not invalidate the schedulability of other tasks. This can be done using the famous theorem by Lawler and Moore. If the selected optional part O_i is schedulable and it also turned out that O_i does not affect the schedulability of other tasks, we schedule it and set c_i to $m_i + o_i$, Otherwise, we reject it.

At the end of this step, we get a new program graph G_1 and L_1. If the selected optional part can be scheduled, then G_1 becomes $G_m \cup \{O_i\}$, otherwise the graph leaves to be G_m. We update L_1 with $L_1 = L - \{O_i\}$. Clearly G_1 is a sub-graph of the initial program graph G.

Once a new program graph G_1 is obtained, we perform the same step again with G_1 instead of G_m. We can apply this procedure repeatedly producing G_2 and L_2, G_3 and L_3, and so on. The algorithm surely terminates because the size of the list L will decease in each step. Figure 4 describes the algorithm in detail.

Theorem : *The algorithm in figure 4 terminates and the graph produced by the algorithm is schedulable.*

Prrof. : The algorithm in figure 4 terminates eventually because the number of optional parts in the list is finite and decreases in each step of the *'while'* loop. The schedulabilty is also guaranteed, since it is checked in every step of *'while'* loop in which an

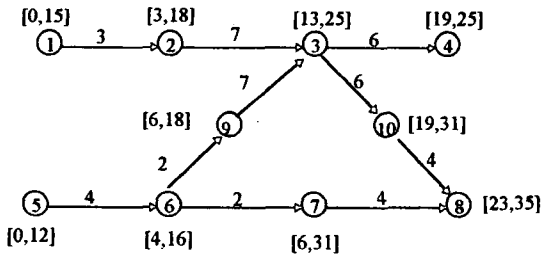


Figure 5.a. Initial program graph G_m for scheduling G in figure

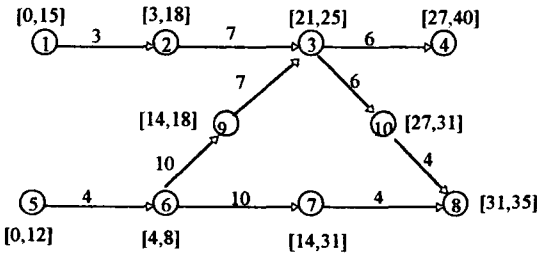


Figure 5.b. Result of scheduling O_2

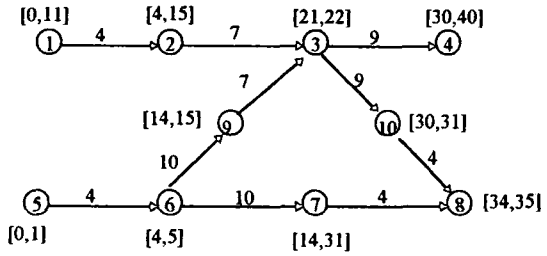


Figure 5.c. Final graph scheduled

Fig. 5. Scheduling of the program graph shown in Fig. 1.

optional part selected from the list is added to the graph.

Figure 5 demonstrates the procedure of scheduling the program graph shown in figure 1. Initially the list L_1 is equal to $\{O_1, O_2, O_3, O_4, O_5, O_6\}$. Modified ready times and deadlines are calculated as in figure 5.a. Optional part O_2 with the largest computational requirement is selected first. Its schedulability is successfully checked and program graph becomes as in figure 5.b. Note that modified ready times and deadlines of tasks 1,2,4,5 and 6 are reevaluated as in figure 5.b. By repeating this procedure for O_4,O_5,O_1,O_6, and O_3, schedule shown in figure 5.c is obtained. Optional part O_2,O_6,and O_3 are schedulable, but O_1,O_4, and O_5 are not. The total error becomes twenty which is sum of the computational requirements of O_1,O_4 and O_5.

Ⅳ. Simulation Results

In this section, we present and analyze the results of the simulation. The aim of simulation is to compare the performance

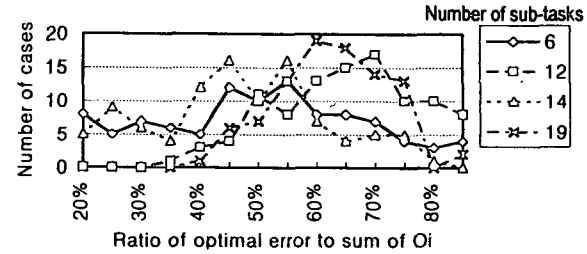


Fig. 6. Distribution of (Optimal Error/Sum of o_i).

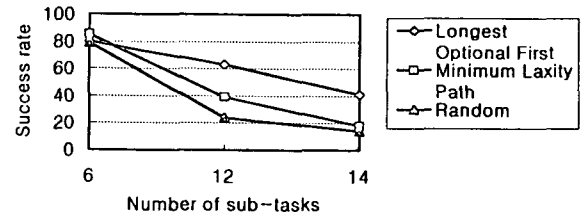


Fig. 7. Distribution of the number of cases such that the total error is equal to optimal error.

of the scheduling algorithm and the selection strategies of optional parts. The sample models of the system under simulation are chosen from the simple control application in[4]. The system described in[4] consists of five tasks such as sensors, actuators, and main control activity that are running on four processors connected by a network. The program graph consists of twenty eight nodes and thirty two edges. The model adopted for simulation is a sub-graph of the program graph in[4]. The number of nodes and edges are chosen according to the number of tasks. However we need extend the tasks so as to have mandatory and optional parts, because the sample by Natale is not designed for the imprecise computation. The values from 1 to 10 are randomly assigned to the computation requirements of mandatory parts and optional parts. The ready times and deadlines of the starting node and ending nodes are fixed throughout the simulation of each model.

Four sample models of the systems were considered: the simplest one has only 6 tasks, and the others have 12, 14 or 19 tasks, respectively. For each model of the system, we generate randomly (100 * number of tasks) pairs of integers from 1 through 10 for the computation requirements of the mandatory and optional parts of each task. Figure 6 shows the distribution of (Optimal Error / Sum of o_i) for each model.

For each system, we run the simulation with the algorithm in figure 4. We consider three different selection strategies. The longest optional first strategy selects the task with the longest optional part among those in the list. The second strategy selects the best fit one in the path with the minimum laxity. The third strategy selects one randomly.

To compare the strategies, we use following metrics. The first metric is defined as a number how about frequently the total error

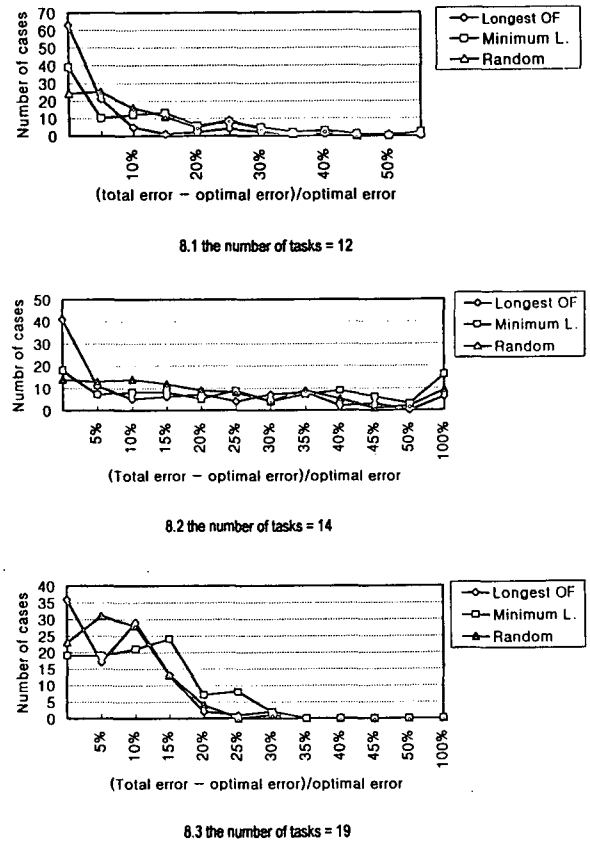


Fig. 8. Distribution of (Total Error-Optimal Error)/Optimal Error.

of the schedule produced by the scheduling algorithm hits the optimal error. The optimal error is calculated by considering all possible combinations of the placement of optional parts. (If the number of optional parts is n, there are 2^n possible cases). Figure 7 shows this distribution. As we can easily note, the longest optional part first strategy hits more frequently than the other two strategies, and the decreasing rate is also much slower than others.

The second metric is the distribution of (Total Error - Optimal Error) / Optimal Error. Figure 8 shows these distributions for the cases that the number of tasks is 12, 14 and 19, respectively. The case with six tasks is omitted because the behavior of three selection strategies did not show any significant difference. As the number of tasks increases, the longest optional part first strategy outperforms the other strategies.

V. Conclusion

We presented a heuristic scheduling algorithms of 0/1 constraint multiprocessor systems that are modeled by directed graphs. The complicated initial graph is reduced to a modified version in [4]. Starting from the program graph, a heuristic algorithm attempts in this paper is applied to find schedulable sub-graph. The total errors depend heavily on the selection strategy by which optional parts are selected. The results of simulations show that the longest optional part first strategy outperforms the random selection strategy or the strategy based on the minimal laxity path.

References

[1] E. A. Hyden, "Operating Support for Quality of Services," Ph.D. thesis, University of Cambridge, 1994.

[2] Liu et al., "Algorithms for Scheduling Imprecise Computations," in Foundation of Real-Time Computing, Scheduling and Resource Management, Kluwer Academic Publishers, 1991.

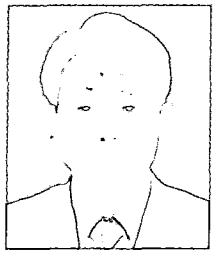
[3] R. McNaughton, "Scheduling with Deadlines and Loss Functions," Management Science, Vol. 12, 1959.

[4] M. C. Natale, J. Stankovic, "Dynamic End-to-End Guarantees in Distributed Real Time Systems," Proc. IEEE Real-Time System Symposium, Dec. 1994.

[5] S. Natarajan, Kenny, K. J.L in, "Building Flexible Real-Time Systems Using the FLEX Language," Computer, Vol. 24, No. 5, May 1991.

[6] D. Niehaus, "Program Representation and Translation to Support Predictability of Real-Time Systems," Proc. IEEE Real-Time Systems Symposium, Dec. 1991.

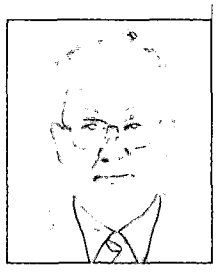
[7] W. K. Shih, J. W. S. Liu et al., "Fast Algorithms for Scheduling Tasks with Ready Times and Deadlines to Minimize Total Error," Proc. IEEE Real-Time Systems Symposium, Dec. 1989.



Ki-Hyun Song was born in Taejon, Korea, on November 23, 1962. He received the B.S. and M.S. degrees in computer science from Chungnam National University, Daejon, Korea, in 1985 and 1987, respectively. He is currently working toward the Ph.D. degree in computer engineering at Ajou University. He is an assistant professor in MIS department at Taejon Medical Junior College. His research interests include real-time task scheduling and natural language processing.

Kyung-Hee Choi has received his B.S. degree from Seoul National University in 1976, his Diplome d'Ingenieur and M.S. degree in the department of Informatics of ENSEEIHT in 1979, and his Ph.D. from Paul Sabatier University in France in 1982. Since 1982, he has joined the division of Information and Computer Engineering at Ajou University. His research interests includes operating systems, distributed systems, multimedia, and real-time systems.

Seung-Kyu Park is a professor in the division of Information and Computer Engineering at Ajou University. Previously, he was at ETRI, Daejon, Korea from 1982 to 1992 and a visiting researcher at the IBM T.J. Watson Research Center from 1984 to 1985. His research interests include multiprocessor architecture, parallel processing, multimedia systems. He earned his B.S. degree from Seoul National University in 1974 and M.S. degree from KAIST in 1976. He earned his Ph.D. degree from Institute National Polytechnique de Grenoble, France in 1982.



Dug-Kyoo Choi is a professor in the division of Information and Computer Engineering at Ajou University. He was with the Agency for Defense Development, as a principal researcher for seventeen years. He received a B.S. degree in Nuclear Engineering from Seoul National University, and his M.S. degree from Wright State University and his Ph.D. degree from University of Massachusetts at Lowell. His research interests include Local Area Network, ATM and the mobile communications.

Kyong-Ok Yun is a senior member of engineering staff in intelligent network service section at ETRI. She received her B.S. and M.S. degree from Chungnam University. Her research interests include Local Area Network and operating systems.