International Journal of Fuzzy Logic and Intelligent Systems, vol. 2, no. 1, March 2002 pp. 59-64

GA-based Adaptive Load Balancing Method in Distributed Systems

Seong Hoon Lee* and Sang Gu Lee**

***Department of Computer Science, Chonan University**
115 Anseo-Dong, Chonan, Choongnam 330-180, Korea
****Department of Computer Engineering, Hannam University**
133 Ojung-Dong, Daeduk-Gu, Taejon 306-791, Korea

Abstract

In the sender-initiated load balancing algorithms, the sender continues to send an unnecessary request message for load transfer until a receiver is found while the system load is heavy. Meanwhile, in the receiver-initiated load balancing algorithms, the receiver continues to send an unnecessary request message for load acquisition until a sender is found while the system load is light. These unnecessary request messages result in inefficient communications, low CPU utilization, and low system throughput in distributed systems. To solve these problems, in this paper, we propose a genetic algorithm based approach for improved sender-initiated and receiver-initiated load balancing. The proposed algorithm is used for new adaptive load balancing approach. Compared with the conventional sender-initiated and receiver-initiated load balancing algorithms, the proposed algorithm decreases the response time and increases the acceptance rate.

Key words : Genetic Algorithm, Load Balancing, Distributed System

Ⅰ. Introduction

Distributed systems consist of a collection of autonomous computers and connected networks. The primary advantages of these system are high performance, availability, and extensibility at low cost. To improve a performance of distributed system, it is essential to keep the system load to each processor equally. An objective of load balancing in distributed systems is to allocate tasks among the processors to maximize the utilization of processors and to minimize the mean response time. Load balancing algorithms can be generally classified into three categories: static, dynamic and adaptive. Our approach is based on the dynamic load balancing algorithm. In dynamic scheme, an overloaded processor (sender) sends excess tasks to an underloaded processor (receiver) during execution. Dynamic load balancing algorithms are specialized into three methods: sender-initiated, receiver-initiated and symmetrically-initiated method. Basically our approach is a sender-initiated and receiver algorithm.

Under sender-initiated algorithms, load balancing activity is initiated by a sender trying to send a task to a receiver[1,2]. In the sender-initiated algorithm, decision of task transfer is made in each processor independently. A request message for the transfer is initially issued from a sender to an another processor randomly selected. If the selected processor is receiver, it returns an accept message. And the receiver is ready for receiving an additional task from sender. Otherwise, it returns a reject message, and the sender tries for others until receiving an accept message. If all the request messages are rejected, no task transfer takes place. While distributed systems remain to light system load, a sender-initiated algorithm performs well. However, when a distributed system becomes to heavy system load, it is difficult to find a suitable receiver because most processors have additional tasks to send. So, many requests and reject messages are repeatedly sent back forth, and a lot of time is consumed before execution.

Similarly, under receiver-initiated algorithm, load balancing activity is initiated by a receiver trying to receive a task from a sender[1,2]. In receiver-initiated algorithm, decision of task acquisition is made in each processor independently. A request message for the task acquisition is initially issued from a receiver to an another processor randomly selected. If the selected processor is sender, it returns an accept message. And the sender is ready for transferring an additional task to receiver. Otherwise, it returns a reject message, and the receiver tries for other until receiving an accept message. If all the request messages are rejected, no task acquisition takes place. while distributed systems remain to heavy system load, a receiver initiated algorithm performs well. However, when a distributed system becomes to light system load, it is difficult to find a suitable sender because most processors have small tasks. So, many request and reject messages are repeatedly sent back and forth, and a lot of time is consumed before execution. Therefore, much of the task processing time is consumed, and causes low system throughput and low CPU utilization.

To solve these problems in sender-initiated and receiver-initiated algorithm, we use a new genetic algorithm and extend to a new adaptive load balancing algorithm. A new genetic

Manuscript received January 14, 2002 ; revised February 20. 2002

algorithm evolves strategy for determining a destination processor to receive a task in sender-initiated algorithm and to send a task in receiver-initiated algorithm. And we define a suitable fitness function. In this scheme, a number of request messages issued before acception a task are determined by the proposed genetic algorithm. This algorithm applies to a population of binary strings. Each gene in the string stands for a number of processors which request messages should be sent off.

II. Genetic Algorithm-based Approach

In this section, we describe various factors to be needed for GA-based load balancing. These are load measure, coding method, fitness function and algorithm.

2.1 Load Measure

We employ the CPU queue length as a suitable load index because this measure is known as the most suitable index[5]. This measure means a number of tasks in CPU queue residing in a processor.

We use a 3-level scheme to represent a load state on its own CPU queue length of a processor. Table 1 shows the 3-level load measurement scheme. Tupp and Tlow are algorithmic design parameters and are called upper and lower thresholds respectively.

Table 1. 3-level load measurement scheme

light-load	CPU queue length $\leq T_{low}$
normal-load	T_{low} < CPU queue length $\leq T_{upp}$
heavy-load	CPU queue length > T_{upp}

The transfer policy uses the threshold policy that makes decisions based on the CPU queue length. The transfer policy is triggered when a task arrives. A node identifies as a sender if a new task origination at the node makes the CPU queue length exceed T_{upp}. A node identifies itself as a suitable receiver for a task acquisition if the node's CPU queue length will not cause to exceed T_{low}.

Each processor in distributed systems has its own population which genetic operators are applied to. There are many encoding methods; Binary encoding, character and real-valued encoding and tree encoding[12]. We use binary encoding method in this parer. So, a string in population can be defined as a binary-coded vector $<v_0, v_1, \ldots, v_{n-1}>$ which indicates a set of processors to which the request messages are sent off. If the request message is transferred to the processor Pi (where $0 \leq i \leq n-1$, n is the total number of processors), then $v_i=1$, otherwise $v_i=0$. Each string has its own fitness value. We select a string by a probability proportional to its fitness value, and transfer the request messages to the processors indicated by the string.

2.2 Sender-based Load Balancing Approach

In the sender-based load balancing approach using genetic algorithm, processors received the request message from the sender send accept message or reject message depending on its own CPU queue length. In the case of more than two accept messages returned, one is selected at random.

Suppose that there are 10 processors in distributed systems, and the processor P_0 is a sender. Then, genetic algorithm is performed to decide a suitable receiver. It is selected a string by a probability proportional to its fitness value. Suppose a selected string is <-,1, 0, 1, 0, 0, 1, 1, 0, 0>, then the sender P0 sends request messages to the processors (P_1, P_3, P_6, P_7). After each processor (P_1, P_3, P_6, P_7) receives a request message from the processor P_0, each processor checks its load state. If the processor P_3 is a light load state, the processor P_3 sends back an accept message to the processor P_0. Then the processor P_0 transfers a task to the processor P_3.

Each string included in a population is evaluated by the fitness function using the following equation in the sender-initiated approach.

$$F_i = 1 / (\alpha \times TMP + \beta \times TMT + \gamma \times TTP)$$

Here, α, β and γ mean the weights for parameters such as TMP,TMT and TTP. The purpose of the weights is to be operated equally for each parameter to fitness function F_i. TMP(Total Message Processing time) is the summation of the processing times for request messages to be transferred. TMT(Total Message Transfer Time) means the summation of each message transfer time from the sender to processors corresponding to bits set '1' in selected string. TTP(Total Task Processing Time) is the summation of the times needed to perform a task at each processor corresponding to bits set '1' in selected string. This algorithm consists of five modules such as initialization, Check_load, String_evaluation, Genetic_operation and Message_evaluation, Genetic_operation module consists of three sub-modules such as Local_improvement_operation, Reproduction and Crossover. These modules are executed at each processor in distributed systems. The algorithm of the proposed method for sender-initiated load balancing is as following.

GA-based sender-initiated load balancing algorithm

```
Procedure Genetic_algorithm Approach

{ Initialization()
    while(Check_load())
      if (Load_i>T_up) {
        Individual_evaluation();
        Genetic_operation();
        Message_evaluation(); }
    Process a task in local processor;
}

Procedure Genetic_operation()
 { Local_improvement_operation();
   Reproduction();
   Crossover();
 }
```

An initialization module is executed in each processor. A population of strings is randomly generated without duplication.

A check_load module is used to observe its own processor's load by checking the CPU queue length, whenever a task is arrived in a processor, if the observed load is heavy, the load balancing algorithm performs the following modules. A individual_evaluation module calculates the fitness value of strings in the population. A Genetic_operation modules such as Local_improvement_operation, Reproduction and Crossover are executed on the population in such a way as follows. Distributed systems consist of groups with autonomous computers. When each group consists of many processors, we can suppose that there are P parts in a string corresponding to the groups. The following genetic operations are applied to each string, and new population of strings is generated:

(1) Local_improvement_Operation

String 1 is chosen, A copy version of the string 1 is generated and part 1 of the newly generated string is mutated. This new string is evaluated by proposed fitness function. If the evaluated value of the new string is higher than that of the original string, replace the original string with the new string. After this, the local improvement of part 2 of string 1 is done repeatedly. This local improvement is applied to each part one by one. When the local improvement of all the parts is finished, new string 1 is generated. String 2 is then chosen, and the above-mentioned local improvement is done. This local_improvement_operation is applied to all the strings in the population.

```
/* Algorithm form local_improvement_operation*/
 for(i=1; i<=total_string_number; i++)
{
   select string[i];
   generated copy version of the selected string[i];
   for (j=1; j<=total_part_number; j++)
       /* total_part_number=p */
       {
         select a part[j] of the copy version;
            apply mutation operator to part[j];
         evaluate the mutated new string;
         if( fitness of new string > fitness of
                    original string)
               original string ← new string;
       }
}
```

(2) Reproduction

The reproduction operation is applied to the newly generated strings. We use the "wheel of fortune" technique[4].

(3) Crossover

The crossover operation is applied to the newly generated strings. These newly generated strings are evaluated. We applied to the "one-point" crossover operator in this paper[4]. One-point crossover used in this paper differs from the pure one-point crossover operator. In the pure one-point crossover, crossover activity generated based on randomly selected crossover point in the string. However, boundaries between parts(p) are used as an alternative of crossover points in this paper. So, we select a boundary among many boundaries at random. And a selected boundary is used as a crossover point. This purpose is to preserve effect of the local_improvement_operation of the previous phase. The crossover activity in this paper is represented as Fig. 1.

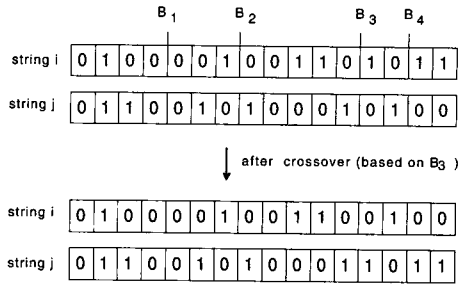

Fig. 1. Crossover activity

Suppose that there are 5 parts in distributed systems. A boundary among the many boundaries ($B_1B_2B_3B_4$) is determined at random as a crossover point. If a boundary B_3 is selected as a crossover point, crossover activity generate based on the B_3. So, the effect of the local_improvement_operation in the previous phase is preserved through crossover activity.

The Genetic_operation selects a string from the population at the probability proportional to its fitness, and then sends off the request messages according to the contents of the selected string.

A message_evaluation module is used whenever a processor receives a message from other processors. When a processor P_i receives a request message, it sends back an accept or reject message depending on its CPU queue length.

2.3 Receiver-based load balancing approach

In the receiver-based load balancing approach, a receiver finds a sender to obtain a task. Processors received the request message from the receiver send accept message or reject message depending on its own CPU queue length. In the case of more than two accept messages returned, one is selected at random.

Suppose that there are 10 processors in distributed systems, and the processor P0 is a receiver, Then, genetic algorithm is performed to decide a suitable sender. It is selected a string by a probability proportional to its fitness value. Suppose a selected string is <-, 1, 0, 1, 0, 0, 1, 1, 0, 0> then the receiver P_0 sends request messages to the processors(P_1, P_3, P_6, P_7). After each processor(P_1, P_3, P_6, P_7) receives a request message from the processor P_0, each processor checks its load state. If the processor P_3 is a heavy load state, the processor P_3 sends back an accept message to the processor P_0. Then the processor P_0 receives a task from the processor P_3.

Each string included in a population is evaluated by the fitness function using the following equation in the receiver-initiated approach.

$$F_i = 1 / (\alpha \times TMP + \beta \times TMT) + \gamma \times TTP$$

So, in order to have a largest fitness value, each parameter such as TMP, TMT must have small values as possible as, but TTP must have large value. That is, TMP must have the

fewer number for request messages, and TMT must have the shortest distance, and TTP should have the larger number of tasks. Eventually, in GA-based receiver-initiated load balancing algorithm, a receiver easily must find a sender(overloaded processor). Therefore, GA evolves towards a fewer number of request messages and shorter distance and larger number of tasks. Eventually, a string with the largest fitness value in population is selected. And after genetic_operation is performed, the request messages are transferred to processors corresponding to bits set '1' in selected string.

In the receiver-based load balancing approach, a receiver finds a suitable sender to obtain a task. Therefore, a receiver performs GA-based load balancing algorithm. Processors received the request message from the receiver send accept message or reject message depending on its own CPU queue length.

GA-based receiver-initiated load balancing algorithm

```
Procedure Genetic_algorithm Approach
  {  Initialization();
     while(Check_load())
       if(Loadi<=Tlow) {
         Individual_evaluation();
         Genetic_operation();
         Message_evaluation(); }
     Process a task in local processor;
  }
Procedure Genetic_operation()
  { Local_improvement_operation();
    Reproduction();
    Crossover();
  }
```

III. Adaptive Load Balancing Approach

The purpose of this paper eventually uses the GA-based sender-initiated and receiver-initiated algorithm for new adaptive load balancing algorithm. The adaptive load balancing algorithm will be performed as the following. If the load of distributed system is low state, it is used a conventional sender-initiated algorithm when a task insert to a specific processor. However, it is used a GA-based receiver-initiated load balancing algorithm when a task go out from a specific processor.

If the load of distributed system is heavy state, it is used a GA-based sender-initiated algorithm when a task insert to a specific processor. But it is used a conventional receiver-initiated load balancing algorithm when a task go out from a specific processor. The algorithm for above described contents is as following.

An Adaptive Load Balancing Algorithm

If (load of distributed system < T_{low})
  If (a specific processor == sender)
use a conventional sender-initiated loadbalancing algorithm
Else use a proposed a GA-based receiver-initiated load balancing algorithm
Else
  If (a specific processor == receiver)
use a conventional receiver-initiated load balancing algorithm
  Else use a proposed a GA-based sender-initiated load balancing algorithm

IV. Experiments

We executed several experiments on the proposed genetic algorithm approach to compare with the conventional sender-initiated and receiver-initiated algorithms. Our experiments have the following assumptions. Firstly, each task size and task type are the same. Secondly, the load rating over the systems is about 60 percent. And the number of part(p) in a string is four. In genetic algorithm, crossover probability(p_c) is 0.7, mutation probability(p_m) is 0.1. The values of these parameters p_c, p_m are known as the most suitable values in various applications[3]. Table 2 shows the detailed contents of the parameters used in our experiments.

Table 2. experimental parameters

number of processors	24
P_c	0.7
P_m	0.1
number of strings	50
number of tasks to be performed	5,000
weight for TMP	0.025
weight for TMT	0.01
weight for TTP	0.02

[Experiment 1] We compared the performance of the proposed method with a conventional method in this experiment by using the parameters on the Table 2. The experiment is to observe change of response time when the number of tasks to be performed is 5,000.

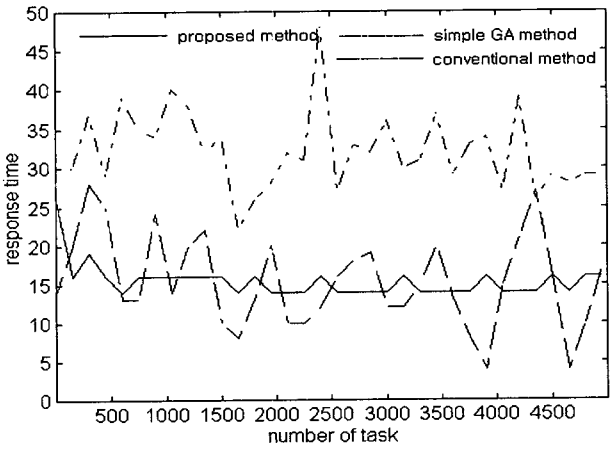

Fig. 2. Results of response time

Fig. 2 shows result of the experiment 1. In conventional methods, when the sender determines a suitable receiver, it select a processor in distributed system randomly, and receive the load state information form the selected processor. The algorithm determines the selected processor as receiver if the load of randomly selected processor is Tlow (light-load). These processes are repeated until a suitable receiver is searched. So, the result of response time shows the severe fluctuation. In the proposed algorithm, the algorithm shows the low response time because the load balancing activity performs the proposed genetic_operation considering load states when it determines a receiver.

[Experiment 2] This experiment is to observe the convergence of the fitness function for the best string in the population corresponding to a specific processor in distributed system. In this experiments, we observed the fact that the processor p6 performs about 550 tasks (550 generations) among 5000 tasks, and the proposed algorithm generally converges through 50 generations. A small scale of the fluctuations displayed in this experiment result from the change of the fitness value for the best string selected through each generation.

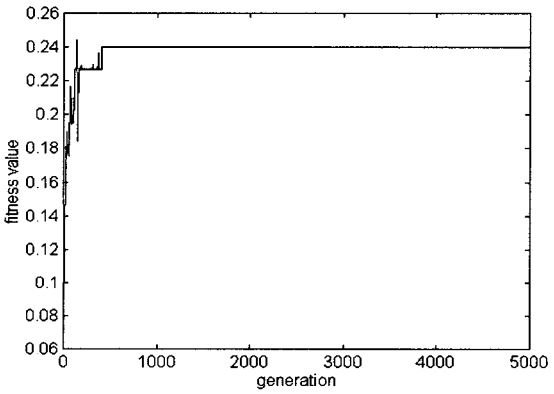

Fig. 3. Fitness value of Processor P6

[Experiment 3] This experiment is to observe the performance when the probability of crossover is changed. Fig. 4 shows the result of response time depending on the changes of Pc when Pm is 0.1. It shows high performances respectively when Pc is 0.7.

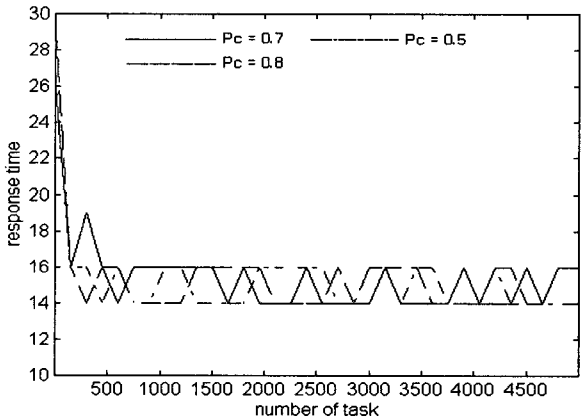

Fig. 4. Results on the change of the Pc

[Experiment 4] This experiment is to observe the performance when the probability of mutation is changed.

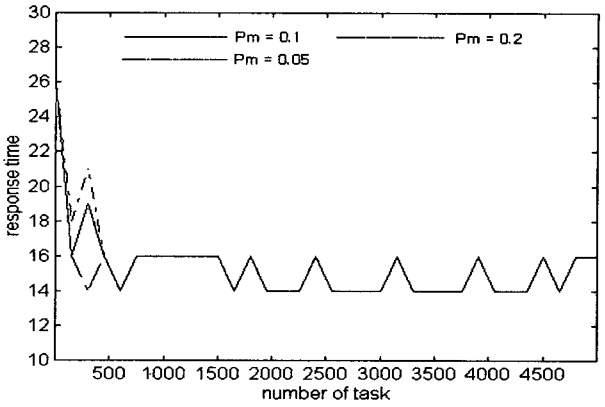

Fig. 5. Results on the change of the Pm

Fig. 5 shows the result of the response time depending on the changes of Pm when Pc is 0.7. The result shows high performances respectively when Pm is 0.1.

[Experiment 5] This experiment is to observe the response time when the system load is 80 %. We know the fact that the proposed algorithm is better than that of conventional sender-initiated algorithm in this experiment.

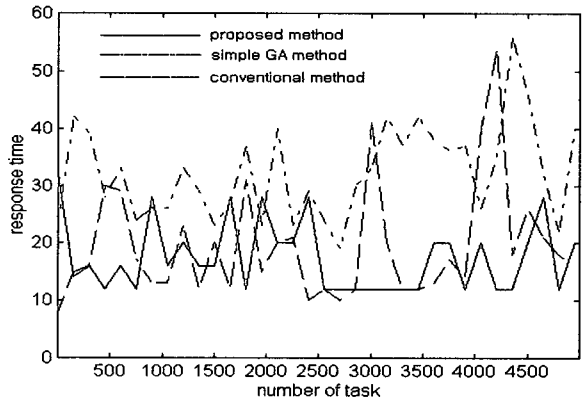

Fig. 6. Response time when system load is 80 %

V. Conclusions

We have proposed a new adaptive load balancing scheme in distributed systems that is based on the genetic algorithm with local improvement operations. The genetic algorithm is used to decide to suitable candidate receivers which task transfer request messages should be sent off. Several experiments have been done to compare the proposed scheme with the conventional schemes on the response time and mean response time. The experimental results show the effectiveness of the proposed algorithm. The performance of the proposed algorithm depending on the changes of the probability of mutation and probability of crossover is also better than that of the conventional schemes. However, the proposed algorithm

is sensitive to the weight values of TMP, TMT and TTP. As a further research, a study on method for releasing sensitivity of weight values is left.

References

[1] D. L. Eager, E. D. Lazowska, J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. on Software Engineering*, vol. 12, no. 5, pp. 662-675, May 1986.

[2] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *IEEE Computer*, vol. 25, no. 12, pp. 33-44, Dec. 1992.

[3] J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Trans. on SMC*, vol. SMC-16, no. 1, pp. 122-128, Jan. 1996.

[4] J. R. Filho and P. C. Treleaven, "Genetic-Algorithm Programming Environments," *IEEE Computer*, pp. 28-43, Jun. 1994.

[5] T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," *IEEE Trans. on Software Engineering*, vol. 17, No. 7, pp. 725-730, Jul. 1991.

[6] T. Furuhashi, K. Nakaoka and Y. Uchikawa, "A New Approach to Genetic Based Machine Learning and an Efficient Finding of Fuzzy Rules" Proc. WWW .94, pp. 114-122, 1994.

[7] J A. Miller, W D. Potter, R V. Gondham and C. N. Lapena, "An Evaluation of Local Improvement Operators for Genetic Algorithms," *IEEE Trans. on SMC*, vol. 23, no. 5, pp. 1340-1451, Sep. 1993.

[8] N. G. Shivaratri and P. Krueger, "Two Adaptive Location Policies for Global Scheduling Algorithms," *Proc. 10th International Conference on Distributed Computing Systems*, pp. 502-509, May 1990.

[9] T. C. Fogarty, F. Vavak and P. Cheng, "Use of the Genetic Algorithm for Load Balancing of Sugar Beet Presses," *Proc. Sixth International Conference on Genetic Algorithms*, pp. 195-624, 1995.

[10] G. W. Greenwood, C. Lang and S. Hurley, "Scheduling Tasks in Real-Time Systems using Evolutionary Strategies," *Proc. Third Workshop on Parallel and Distributed Real-Time Systems*, pp. 195-196, 1995.

[11] M. Srinivas and L. M. Patnait, "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms," *IEEE Trans. on SMC*, vol. 24, no. 4, pp. 656-667, Apr. 1994.

Seong Hoon Lee

He received the B.S. degree in computer science from Hannam University, Taejon, Korea, and the M.S. degree and the Ph.D. degree in Computer Science from Korea University, Korea. Since 1993, he has been a professor in division of Computer Science, Chonan University, Choongnam, Korea. His current research interests are in genetic algorithm, distributed systems and mobile computing.

Phone : +82-41-620-9444
Fax : +82-41-550-9122
E-mail : shlee@mail.chonan.ac.kr

Sang Gu Lee

was born in Seoul, Korea on September 22, 1955. He received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1978, and the M.S. degree in Computer Science from KAIST, Korea in 1981.

He received the Ph.D. degree in Electrical Electronics and Computer Engineering from Waseda University, Tokyo, Japan. From 1996 to 2000, he was a member of international program committee in IASTED. Since 1983, he has been a professor in division of Computer Engineering, Hannam University, Taejon, Korea. His current research interests are in parallel processing, parallel architecture and parallel neuro-fuzzy computing.

Phone : +82-42-629-7551
Fax : +82-42-487-9335
E-mail : sglee@mail.hannam.ac.kr