A Load Sharing Algorithm Including An Improved Response Time using Evolutionary Information in Distributed Systems

SeongHoon Lee

Department of Computer Science
Baekseok University, Cheonan. Korea

ABSTRACT

A load sharing algorithm is one of the important factors in computer system. In sender-initiated load sharing algorithms, 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. The sender continues to send unnecessary request messages for load transfer until a receiver is found while the system load is heavy. Because of these unnecessary request messages it results in inefficient communications, low cpu utilization, and low system throughput. To solve these problems, we propose a self-adjusting evolutionary algorithm for improved sender-initiated load sharing in distributed systems. This algorithm decreases response time and increases acceptance rate. Compared with the conventional sender-initiated load sharing algorithms, we show that the proposed algorithm performs better.

KeyWords: Evolutionary Information, Response Time, Distributed System.

1. INTRODUCTION

Distributed systems consist of a collection of autonomous computers connected network. The primary advantages of these systems are high performance, availability, and extensibility at low cost. To improve a performance of distributed systems, it is essential to keep the system load to each processor equally.

An objective of load sharing in distributed systems is to allocate tasks among the processors to maximize the utilization of processors and to minimize the mean response time. Load sharing algorithms can be largely classified into three classes: static, dynamic, and adaptive. Our approach is based on the dynamic load sharing algorithm. In dynamic scheme, an overloaded processor(sender) sends excess tasks to an underloaded processor(receiver) during execution.

Dynamic load sharing algorithms are specialized into three methods: sender-initiated, receiver-initiated, symmetrically-initiated. Basically our approach is a sender-initiated algorithm.

Under sender-initiated algorithms, load sharing activity is initiated by a sender trying to send a task to a receiver[1],[2]. In sender-initiated algorithm, decision of task transfer is made in each processor independently. A request message for the task 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 to receive 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. But 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 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, low cpu utilization

To solve these problems in sender-initiated algorithm, we use a new evolutionary algorithm. A new evolutionary algorithm evolves strategy for determining a destination processor to receive a task in sender-initiated algorithm. In this scheme, a number of request messages issued before accepting a task are determined by proposed evolutionary algorithm. The proposed evolutionary 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.

The rest of the paper is organized as follows. Section 2 presents the Evolutionary Algorithm-based sender-initiated approach. Section 3 presents several experiments to compare with conventional method. Finally the conclusions are presented in Section 4.

* *Corresponding author. E-mail : shlee@bu.ac.kr*
Manuscript received May. 15, 2008 ; accepted Jun. 20, 2008

2. EVOLUTIONARY ALGORITHM-BASED APPROACH

In this section, we describe various factors to be needed for EA-based load sharing. That is, load measure, representation 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 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. T_{up} and T_{low} are algorithm design parameters, called *upper* and *lower thresholds* respectively.

Table 1. 3-level load measurement scheme

Load state	Meaning	Criteria
L-load	light-load	CQL$\leq T_{low}$
N-load	normal-load	T_{low} < CQL$\leq T_{up}$
H-load	heavy-load	CQL > T_{up}

(CQL : CPU Queue Length)

The transfer policy use the threshold 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 originating at the node makes the CPU queue length exceed T_{up}. 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}.

2.2 Representation

Each processor in distributed systems has its own population which evolutionary 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 paper. So, a string in population can be defined as a binary-coded vector $<v_o, v_1, ..., v_{n-1}>$ which indicates a set of processors to which the request messages are sent off. If the request message is transferred to the processor P_i(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. When ten processors exist in distributed system, the representation is displayed as Fig. 1.

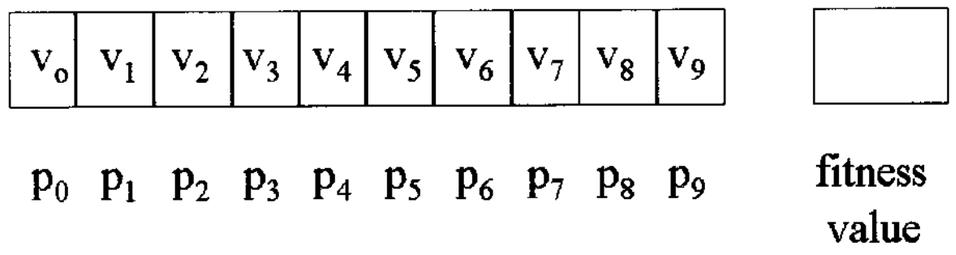

Fig. 1. Representation for processors

2.3 Load Sharing Approach

2.3.1 Overview

In sender-based load sharing approach using evolutionary algorithm, Processors received the request message from the sender send accept message or reject message depending on its own CPU queue length. In 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, evolutionary 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 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 P3 is a light load state, the processor P_3 sends back an accept message to the processor P_0. Then the processor P0 transfers a task to the processor P_3.

2.3.2 Fitness Function

Each string included in a population is evaluated by the fitness function using following formula in sender-initiated approach. α, β, γ used above formula mean the weights for parameters such as *TMP*, *TMT*, *TTP*. The purpose of the weights is to be operated equally for each parameter to fitness function F_i.
Firstly, *TMP*(Total Message Processing time) is the summation of the processing times for request messages to be transferred. This parameter is defined by the following formula. The *ReMN* is the number of messages to be transferred. It means the

$$F_i = \left(\frac{1}{\alpha \times TMP + \beta \times TMT + \gamma \times TTP} \right)$$

$$TMP = \sum_{k \in x} \left(\mathrm{Re}\, MN_k \times Time\ Unit \right)$$

number of bits set '1' in selected string. The objective of this parameter is to select a string with the fewest number of messages to be transferred.

(where, x={i | v_i=1 for $0 \leq i \leq n-1$})

Secondly, *TMT*(Total Message Transfer time) means the summation of each message transfer times(*EMTT*) from the sender to processors corresponding to bits set '1' in selected string. The objective of this parameter is to select a string with the shortest distance eventually. So, we define the TMT as the

following formula.

$$TMT = \sum_{k \in x} EMTT_k$$

(where $x=\{i \mid v_i=1 \quad \text{for } 0 \le i \le n-1\}$)

Last, *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 parameter is defined by the following formula. The objective of this parameter is to select a string with the fewest loads. Load in parameter *TTP* is the

$$TTP = \sum_{k \in x} (Load_k \times TimeUnit)$$

volume of CPU queue length in the processor.

(where $x=\{i \mid v_i=1 \quad \text{for } 0 \le i \le n-1\}$)

So, in order to have a largest fitness value, each parameter such as *TMP*, *TMT*, *TTP* must have small values as possible as. That is, *TMP* must have the fewer number of request messages, and *TMT* must have the shortest distance, and *TTP* should have the fewer number of tasks.

Eventually, a string with the largest fitness value in population is selected. And after evolutionary_operation is performed, the request messages are transferred to processors corresponding to bits set '1' in selected string.

2.3.3 Algorithm

This algorithm consists of five modules such as Initialization, Check_load, String_evaluation, Evolutionary_operation and Message_evaluation. Evolutionary_operation module consists of three sub-modules which are Local_improvement_operation, Reproduction, Crossover. These modules are executed at each processor in distributed systems.

The algorithm of the proposed method for sender-initiated load sharing is presented as Fig. 2.

Algorithm : EA-based sender-initiated load sharing

```
Procedure Evolutionary_Algorithm Approach
 {  Initialization();
    while ( Check_load() )
        if ( Load_i > T_up )  {
            Individual_evaluation();
            Evolutionary_operation();
            Message_evaluation();  }
      Process a task in local processor;
  }
Procedure Evolutionary_operation()
 { Local_improvement_operation();
   Reproduction();
   Crossover();
  }
```

Fig. 2 Proposed algorithm

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 sharing algorithm performs the following modules.

A Individual_evaluation module calculates the fitness value of strings in the population.

A Evolutionary_operation module such as Local_improvement_operation, Reproduction, Crossover is 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 evolutionary 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 population.

```
/* Algorithms for local_improvement_operation */

for (i=1; i<=total_string_number; i++)
{
    select string[i];
    generate 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 pure one-point crossover, crossover activity generates based on randomly selected crossover point in the string. But 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 an effect of the Local_improvement_operation of the previous phase. Therefore, the crossover activity in this paper is represented as Fig. 3.

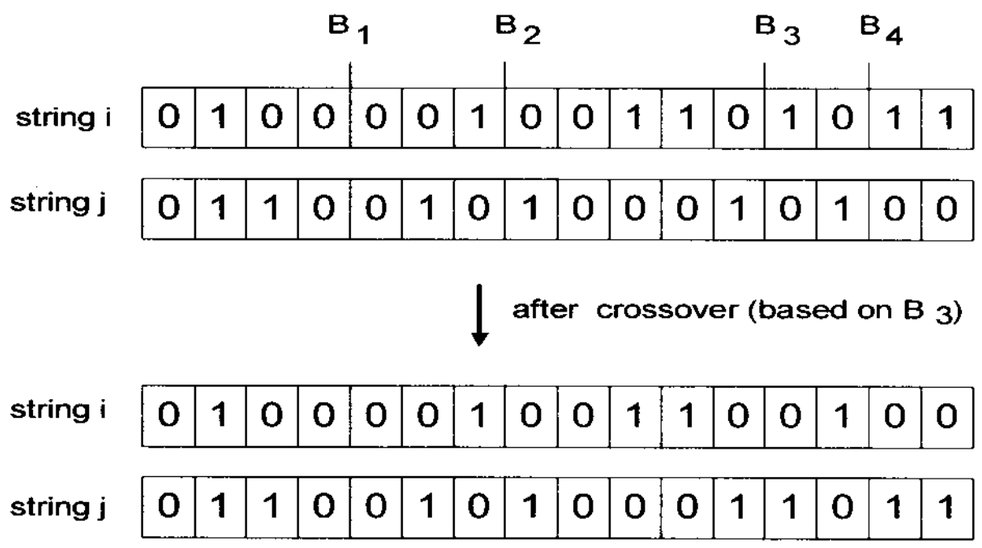

Fig. 3. Crossover Activity

Suppose that there are 5 parts in distributed systems. A boundary among the many boundaries(B_1, B_2, B_3, B_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 Evolutionary_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.

3. EXPERIMENTS

We executed several experiments on the proposed evolutionary algorithm approach to compare with a conventional sender-initiated algorithm

Our experiments have the following assumptions. Firstly, each task size and task type are the same. Secondly, the number of parts(p) in a string is four. In evolutionary algorithm, crossover probability(P_c) is 0.7, mutation probability(P_m) is 0.1. The values of these parameters P_c, P_m were known as the most suitable values in various applications[3]. Table 2 shows the detailed contents of parameters used in our experiments.

Table 2. Contents of parameter

number of processor	24
Pc	0.7
Pm	0.1
number of strings	50
number of tasks to be performed	5000

The parameters and values for fitness value of sender-initiated load sharing algorithm are the same as the table 3. The load rating over systems supposed about 60 percent.

Table 3. Weight values for *TMP*, *TMT* and *TTP*

Weights for *TMP*	0.025
Weights for *TMT*	0.01
Weights for *TTP*	0.02

[Experiment 1] We compared the performance of proposed method with a conventional method in this experiment by using the parameters on the table 2 and table 3. The experiment is to observe change of response time when the number of tasks to be performed is 5000.

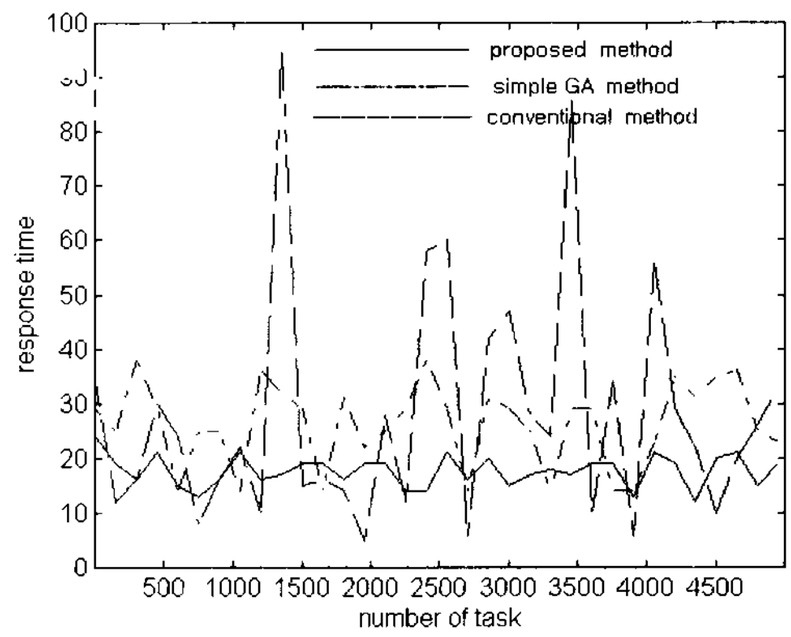

Fig. 4. Result of response time

Fig. 4 shows result of the experiment 1. In conventional methods, when the sender determines a suitable receiver, it select a processor in distributed systems randomly, and receive the load state information from the selected processor. The algorithm determines the selected processor as receiver if the load of randomly selected processor is T_{low}(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 sharing activity performs the proposed evolutionary_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 systems.

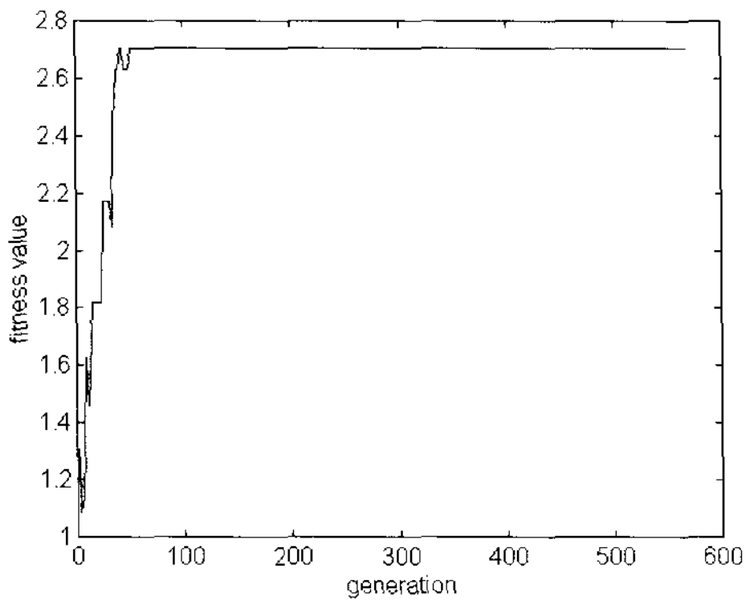

Fig. 5. Fitness value of the processor P_6

In this experiments, we observed the fact that the processor P_6 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.

[Experiment 3] This experiment is to observe the performance when the probability of crossover is changed.

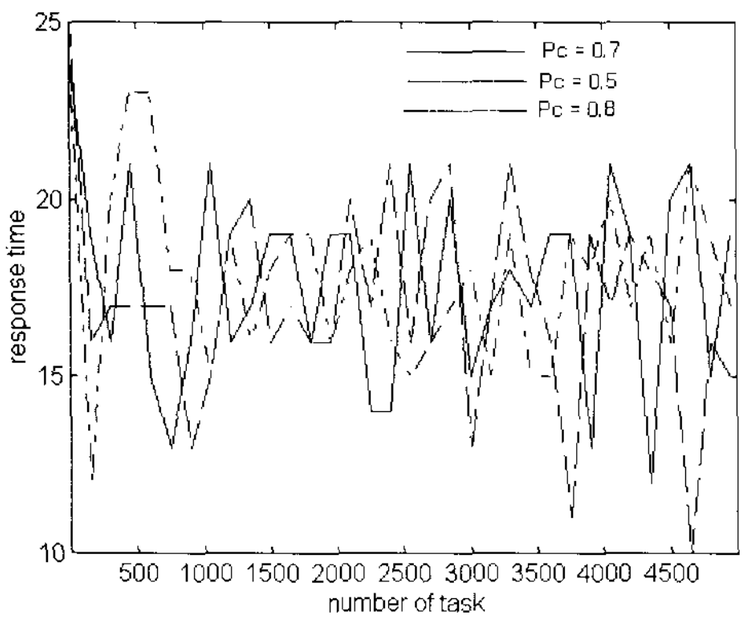

Fig. 6. Result depending on the changes of P_c

Fig. 6 shows the result of response time depending on the changes of P_c when P_m is 0.1. In accordance with value of P_c, It shows a different performance. But the proposed algorithm shows better performance than that of conventional algorithm and simple evolutionary algorithm approach.

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

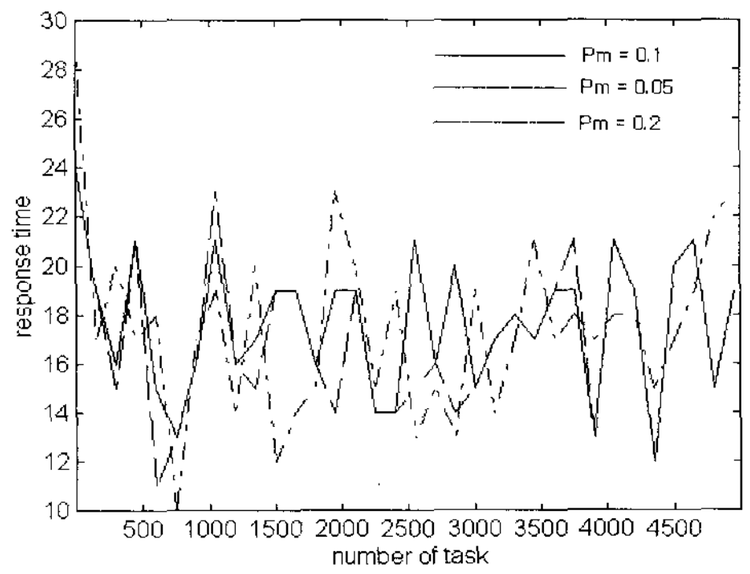

Fig. 7. Result depending on the changes of P_m

Fig. 7 shows the result of the response time depending on the changes of P_m when P_c is 0.7. In accordance with value of P_m, It shows a different performance. But the proposed algorithm shows better performance than that of conventional algorithm and simple evolutionary algorithm approach.

[Experiment 5] This experiment is to observe the response time when the system load is 80percentage. The performance of proposed algorithm is better than that of the conventional algorithm and simple GA algorithm.

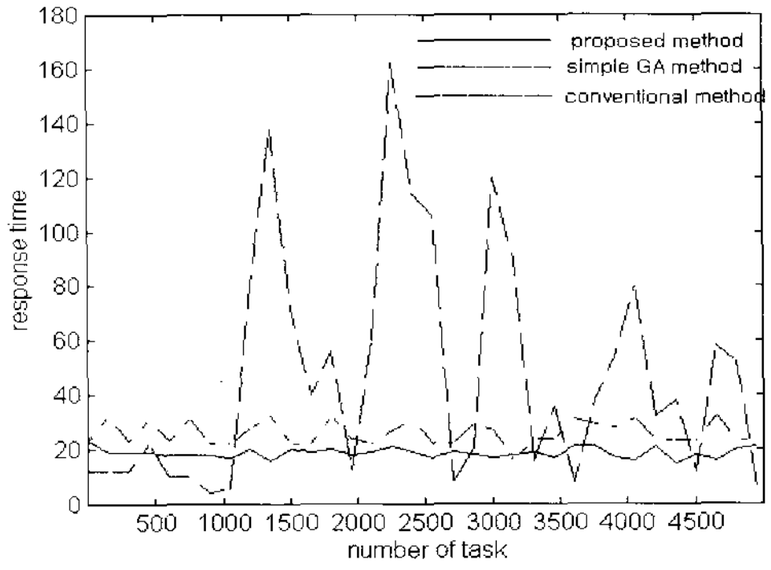

Fig. 8. Response time when system load is 80%

4. CONCLUSIONS

We propose new dynamic load sharing scheme in distributed system that is based on the new evolutionary algorithm with a local improvement operation. The proposed evolutionary 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 a conventional algorithm and simple evolutionary algorithm approach. Through the various experiments, the performances of the proposed scheme is better than that of the conventional scheme and simple evolutionary algorithm approach on the response time and mean response time. The performance of the proposed algorithm depending on

the changes of the probability of mutation(P_m) and probability of crossover(P_c) is also better than that of the conventional scheme and simple evolutionary algorithm approach. But the proposed algorithm is sensitive to the weight values of *TMP*, *TMT* and *TTP*. In future, we will study on method for releasing sensitivity of weight values.

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, May 1986, pp.662-675.

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

[3] J.Grefenstette, *"Optimization of Control Parameters for Genetic Algorithms,"* IEEE Trans on SMC, vol.SMC-16, no.1, January 1986, pp.122-128.

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

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

[6] T.Furuhashi, K.Nakaoka, Y.Uchikawa, *"A New Approach to Genetic Based Machine Learning and an Efficient Finding of Fuzzy Rules,"* Proc. WWW'94, 1994, pp.114-122,.

[7] J A. Miller, W D. Potter, R V. Gondham, C N. Lapena, *"An Evaluation of Local Improvement Operators for Genetic Algorithms,"* IEEE Trans on SMC, vol.23, No 5, Sept 1993, pp.1340-1351.

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

[9] Terence C. Fogarty, Frank Vavak, and Phillip Cheng, *"Use of the Genetic Algorithm for Load Balancing of Sugar Beet Presses,"* Proc. Sixth International Conference on Genetic Algorithms, 1995, pp.617-624.

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

[11] Gilbert Syswerda, Jeff Palmucci, *"The application of Genetic Algorithms to Resource Scheduling,"* Proc. Fourth International Conference on Genetic Algorithms, 1991, pp.502-508.

[12] Melanie Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996.

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

Seong Hoon Lee

He received 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.