An Intelligent New Dynamic Load Redistribution Mechanism in Distributed Environments

Seonghoon Lee
Department of Computer Science
Baekseok University, Cheonan, Korea

ABSTRACT

Load redistribution is a critical resource in computer system. In sender-initiated load redistribution algorithms, the sender continues to send unnecessary request messages for load transfer until a receiver is found while the system load is heavy. These unnecessary request messages result in inefficient communications, low CPU utilization, and low system throughput in distributed systems. To solve these problems, we propose a genetic algorithm based approach for improved sender-initiated load redistribution in distributed systems. Compared with the conventional sender-initiated algorithms, the proposed algorithm decreases the response time and task processing time.

***Keywords**: Load Distribution, Genetic Algorithm, Distributed Environments.*

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 redistribution in distributed systems is to allocate tasks among the processors to maximize the utilization of processors and to minimize the mean response time. Load redistribution algorithms can be largely classified into three classes: static, dynamic[1-2][8], adaptive[9]. Our approach is based on dynamic load redistribution algorithm. In dynamic scheme, an overloaded processor(sender) sends excess tasks to an underloaded processor(receiver) during execution.

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

Under sender-initiated algorithms, load redistribution activity is initiated by a sender trying to send a task to a receiver [1-2][8]. Decision of task transfer is made in each processor independently. A request message for the task transfer is initially issued from a sender to 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. While distributed systems remain to light system load, the sender-initiated algorithm performs well. But when the distributed systems become 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 genetic algorithm. The new genetic algorithm evolves strategy for determining a destination processor to receive a task in sender-initiated algorithm. And we define a suitable fitness function. In this scheme, a number of request messages issued before accepting a task are determined through the proposed genetic algorithm. The proposed genetic 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.

2. RELATED WORKS

In study[7], Evolutionary algorithm was showed to be adapted in scheduling problems. In study[8], The authors compared with different properties for many load redistribution algorithms. The different properties in [8] contained such as load measurement, information exchange, load redistribution operation, genetic algorithms.

3. INTELLIGENT GA-BASED LOAD REDISTRIBUTION METHOD

In this section, we describe various factors to be needed for

* *Corresponding author. E-mail : shlee@bu.ac.kr*
Manuscript received Mar. 08, 2007 ; accepted Mar. 27, 2007

GA-based load redistribution. These are load measure, coding method, fitness function and algorithm.

3.1 Load Measure

We employ the CPU queue length as a suitable load index because this measure is known as a 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 and are called upper and lower thresholds respectively.

Transfer policy uses the threshold policy that makes decision based on 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}.

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)

3.2 Coding Method

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, Tree encoding. We use binary encoding method in this paper. So, a string in population can be defined as a binary-coded vector $<v_o,v_1,\ldots,v_{n-1}>$ which indicates a set of processors to which the request messages are sent off. If the request message is transferred to the processor 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.

3.3 Fitness Function (String evaluation)

In the sender-based load redistribution 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 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 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 following formula.

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

Here, α, β, γ 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.

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 times 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. *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. In experiment, task processing time in each processor was obtained by the random number generator. The objective of this parameter is to select a string with the fewest loads. 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.

3.4 Algorithm

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, Reproduction, Crossover. These modules are executed at each processor in distributed systems. The algorithm of the proposed sender-based load redistribution is as following.

```
/* GA-based sender-initiated load redistribution algorithm */
    Procedure Genetic_algorithm Approach

      { Initialization()
          while (Check_load())
              if (Loadi > Tup) {
                  String_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 redistribution algorithm performs the above modules. A String_evaluation module calculates the fitness value of strings in the population by using F_i. A Genetic_operation module such as Local improvement, Reproduction, Crossover is executed on the population in such a way as above. 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 strings, 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 effect of the local improvement operation of the previous phase.

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 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.

4. EXPERIMENTS

We executed several experiments on the proposed genetic algorithm approach to compare with the conventional sender-initiated algorithm. Our experiments have the following assumptions. First, each task size and task type are the same. Second, the number of parts(p) in a string is five. In genetic algorithm, crossover probability(P_c) is 0.7, mutation probability(P_m) is 0.05. The values of these parameters P_c, P_m were known as the most suitable values in various applications[3][6]. The table 2 shows the detailed contents of parameters used in our experiments.

Table 2. experimental parameters

number of processor	30
P_c	0.7
P_m	0.05
number of strings	50

Our experiments for sender-initiated load redistribution approach have the following assumptions. The load rating over the systems is about 60 percent. The number of tasks to be performed is 3000. The weight for *TMP* is 0.025. The weight for *TMT* is 0.01. The weight for *TTP* is 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. The experiment is to observe the change of response time when the number of tasks to be performed is 3000.

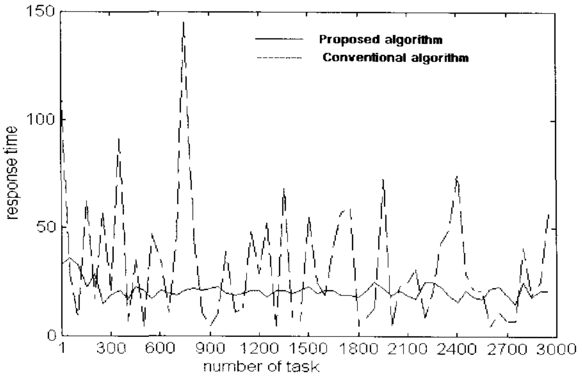

Fig. 1. Result of response time

Fig. 1 shows result of the experiment 1. In conventional method, 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 redistribution activity performs the genetic_operation considering the load state when it determines a receiver.

[Experiment 2] These experiments are to observe the performance when the probability of crossover is changed and is to observe the performance when the probability of mutation is changed.

Fig. 2 shows the result of response time depending on the changes of P_c when P_m is 0.05. It shows high performance respectively when P_c is 0.7. Fig. 3 shows the result of the response time depending on the changes of P_m when P_c is 0.7. The result shows high performance respectively when P_m is 0.1.

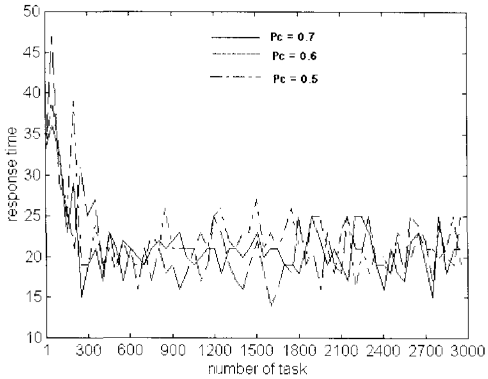

Fig. 2. Result depending on the changes of P_c

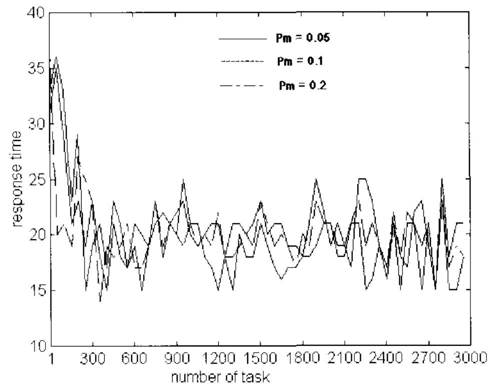

Fig. 3. Result depending on the changes of P_m

[Experiment 3] We compared the performance of proposed method with a conventional method and simple genetic algorithm based method. The experiment is to observe the change of response time when the number of tasks to be performed is 5000.

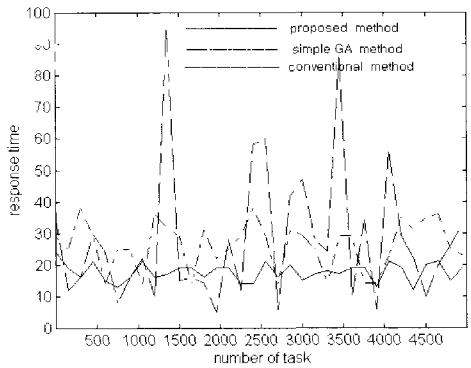

Fig. 4. Result of response time

Fig. 4 shows the result of response time for each algorithm. The conventional method shows severe fluctuation because of randomness. But the proposed method shows the low fluctuation because of the proposed GA-based load redistribution activity.

5. CONCLUSIONS

We propose a new load sharing scheme in distributed system that is based on the genetic algorithm. The genetic algorithm is used to decide to suitable candidate senders or receiver receivers which task transfer request messages should be sent off. Several experiments have been done to compare the proposed scheme with a conventional algorithm. Through the various experiments, performances of proposed scheme are better than those of conventional scheme on the response time and mean response time. 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, December 1992.

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

[4] J.R. Filho and P. C. Treleaven, "Genetic-Algorithm Programming Environments," *IEEE COMPUTER*, pp.28-43, June 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, July 1991.

[6] 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, pp.1340-1351, Sept 1993.

[7] 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*, pp.195-196, 1995.

[8] A Y. Zomaya, Y H. Teh, "Observations on using Genetic Algorithms for Dynamic Load-Balancing," IEEE Trans on Parallel and Distributed Systems, vol.12, No 9, pp.899-911, Sept 2001.

[9] S H. Lee, D W Lee, W K Lee, H J Cho, "An Adaptive Load Balancing Approach in Distributed Computing Using Genetic Theory," LNCS3320, pp 322~325, 2004.

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.