An Enhanced Response Time Mechanism in Grid Systems

SeongHoon Lee
Communication & Information Division
Baekseok University, Cheonan City, Choongnam, Korea

ABSTRACT

For applications that are grid enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization. When jobs communicate with each other, the internet, or with storage resources, an advanced scheduler could schedule them to minimize communications traffic or minimize the distance of the communications. We propose an intelligent load distribution algorithm to minimize communications traffic and distance of the communications using genetic algorithm. The experiments show the proposed load redistribution algorithm performs efficiently in the variance of load in grid environments.

***Keywords**: Grid Systems, Resource Management, Scheduling, Intelligent Load Distribution.*

1. INTRODUCTION

A grid federates a large number of resources contributed by individual machines into a large single-system image. For applications that are grid enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization[1, 2, 3, 5]. This feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization. This can happen in the ways: An unexpected peak can be routed to relatively idle machines in the grid[2, 3, 5]. Other more subtle benefits can occur using a grid for load redistribution. When jobs communicate with each other, the internet, or with storage resources, an advanced scheduler could schedule them to minimize communications traffic or minimize the distance of the communications. This can potentially reduce communication and other forms of contention in the grid. An objective of load redistribution in grid environment is to allocate tasks among the processors to maximize the utilization of processors and to minimize the mean response time. In a dynamic scheme, an overloaded processor(sender) sends excess tasks to an underloaded processor(receiver) during execution.

Load redistribution activity is initiated by a sender trying to send a task to a receiver in sender-based systems[7, 8, 9]. 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 the sender. Otherwise, it returns a reject message, and the sender tries for others until receiving an accept message. When the grid environment becomes 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.

To solve these problems in sender-based algorithm, we use a new genetic algorithm. In this scheme, a number of request messages issued before accepting a task are determined through a proposed genetic algorithm.

2. PROPOSED ALGORITHM

2.1 Factors to be considered

We employ the CPU queue length as a suitable load index because this measure is known as a most suitable index[8]. 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. In 3-level 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(CQL). 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}.

Each processor in grid environment has its own population which genetic operators are applied to. There are many encoding methods. 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 $0 \le i \le n-1$, n is the total number of processors), then $v_i=1$, otherwise $v_i=0$.

* *Corresponding author. E-mail : shlee@bu.ac.kr*
Manuscript received May. 03, 2009 ; accepted Jun. 22, 2010

2.2 Algorithm Approach

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. Each string included in a population is evaluated by the fitness function using following formula.

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

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. 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.

The Fig. 1 shows our proposed general algorithm.

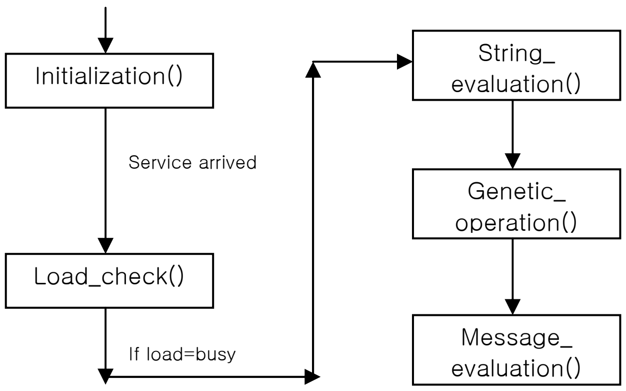

Fig. 1. General Algorithm

This algorithm consists of five modules such as *Initialization*, *Load_check*, *String_evaluation*, *Genetic_operation* and *Message_evaluation*. These modules are executed at each processor in grid environment. The algorithm of the proposed sender-based load redistribution is as following.

```
{ Initialization()
  while (Load_check())
    if (Load$i$ > $T_{up}$) {
      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 *Load_check* 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. A *Genetic_operation* module such as *Local improvement*, *Reproduction*, *Crossover* is executed on the population in such a way as above. Grid environment consists 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:

In *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.

In *reproduction(),* the reproduction operation is applied to the newly generated strings. We use the "wheel of fortune" technique[6].

In *crossover(),* the crossover operation is applied to the newly generated strings. These newly generated strings are evaluated. 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. So we select a boundary among many boundaries at random. And a selected boundary is used as a crossover point. Its purpose is to preserve effect of the local improvement operation of the previous phase.

Suppose that there are 5 parts in grid environment. 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.

Suppose that there are 10 processors in distributed systems, and the processor P_2 is a sender. Then, genetic algorithm is performed to decide a suitable receiver. And it is selected a string by a probability proportional to its fitness value. Suppose a selected string S_5 is <1, 0, -, 1, 1, 0, 1, 0, 1, 0>, then the sender P_2 sends request messages to the processors (P_0, P_3, P_4, P_6, P_8). After each processor(P_0, P_3, P_4, P_6, P_8) receives a request message from the processor P_2, each processor checks its load state. If the processor P_0 is a light load state, the processor P_0 sends back an accept message to the processor P_2. Then the processor P_2 transfers a task to the processor P_0.

Applied Genetic_operation()

↓

(Example for processor P_2)

String No	String content	Fitness value
S_1	0, 0, −, 0, 1, 0, 0, 0, 0, 0	0.1
S_2	1, 0, −, 0, 0, 0, 0, 0, 0, 1	0.2
S_3	0, 0, −, 1, 0, 1, 1, 0, 0, 1	0.5
S_4	0, 0, −, 0, 1, 0, 0, 1, 0, 1	0.3
S_5	1, 0, −, 1, 1, 0, 1, 0, 1, 0	0.9
S_6	1, 0, −, 0, 0, 1, 0, 1, 0, 0	0.6
S_7	0, 1, −, 1, 1, 0, 0, 0, 0, 1	0.4
S_8	0, 1, −, 0, 1, 0, 1, 0, 0, 1	0.6
S_9	1, 0, −, 0, 1, 0, 0, 1, 1, 0	0.8

↓

String selection

↓

Select S_5 (fitness value highest)

↓

Request service to P_0, P_3, P_4, P_6, P_8

Fig. 2. Algorithm processing flow

3. EXPERIMENTS AND ANALYSIS

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. The values of these parameters P_c, P_m were known as the most suitable values in various applications[6]. The table 1 shows the detailed contents of parameters used in our experiments. The load rating over the systems is about 60 percent. The number of tasks to be performed is 3000.

Table 1. Experimental parameters

Number of processor	24
Pc	0.7
Pm	0.1
Number of strings	60
Weight of TMP	0.02
Weight of TMT	0.01
Weight of 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 1. The experiment is to observe the change of response time..

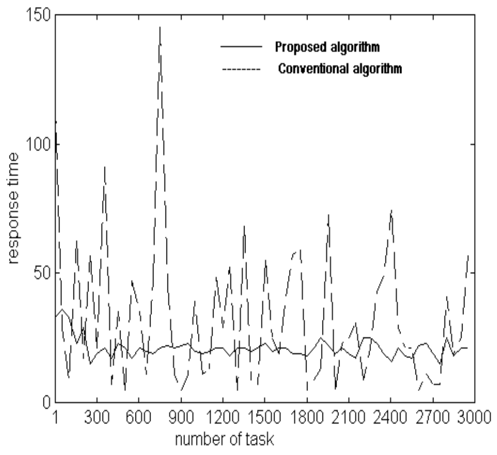

Fig. 3. Result of response time

Fig 3 shows result of the experiment 1. In conventional method, when the sender determines a suitable receiver, it selects a processor in grid environment randomly, and receives 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_{lo}. 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 are to observe the performance when the probability of mutation is changed.

Fig 4 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 5 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.

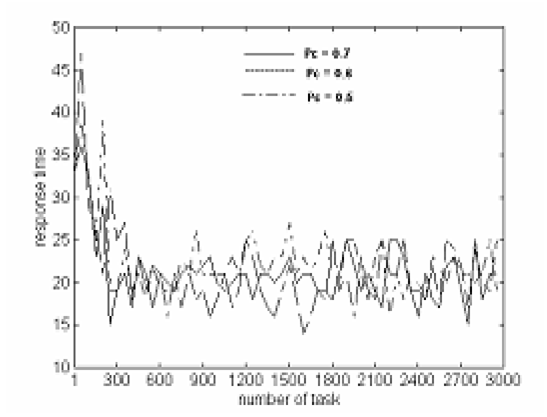

Fig. 4. Result depending on the changes of P_c

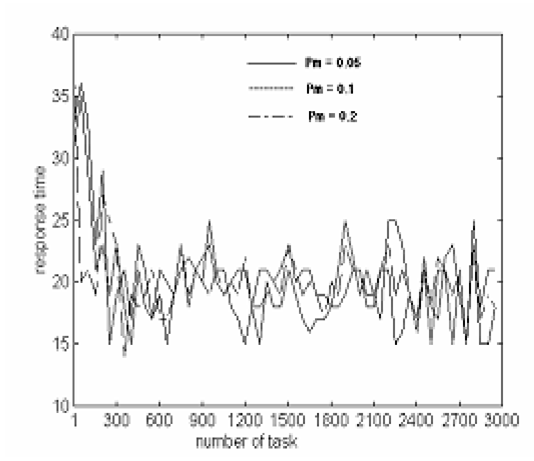

Fig. 5. Result depending on the changes of P_m

Through these several experiments, we verify that the performance of the proposed scheme is better than that of the conventional scheme. The performance depending on the changes for parameters P_c and P_m is excellent than that of the conventional method.

4. CONCLUSIONS

We proposed a new intelligent load redistribution scheme in distributed system that is based on a genetic algorithm. Several experiments have been done to compare the proposed scheme with a conventional algorithm. The various experiments proved that performances of the proposed scheme are better than those of the conventional scheme on in response time and mean response time. However the proposed algorithm is sensitive to the weight values of *TMP*, *TMT* and *TTP*. For a further research, a study on the method for releasing the sensitivity of weight values is suggested.

REFERENCES

[1] B. Jacob, M. Brown., *Introduction to grid computing*. ibm.com/redbooks.
[2] B. Jacob, M. Brown, K. Fukui, "Introduction to Grid Computing," IBM Redbooks, http://ibm.com/redbooks.
[3] Y, Li, Z. Lan, "A Survey of Load Balancing in Grid Computing," LNCS 3314, Dec 2004, pp. 280-285
[4] http://www.gridforum.org.
[5] J. Joseph, C. Fellenstein., *Grid Computing*, IBM Press.
[6] J.Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," IEEE Trans on SMC, vol. SMC-16, no.1, January 1986, pp.122-128.
[7] 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.
[8] 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.
[9] 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.
[10] Thomas G. R., Ten Reasons to Use Divisible Load Theory. IEEE Computer Vol. 36, No. 5, 2003, pp. 63-68.
[11] http://www.cs.wisc.edu/condor.
[12] http://www.gridforumkorea.org.
[13] http://www.globus.org/.

SeongHoon Lee
I received the Ph.D. in computer science from Korea University, KOREA in 1998. Since then, I has been with Baekseok Univ. Main research interests include Distributed system, Grid Computing, Web Service, etc