An Efficient Load-Sharing and Fault-Tolerance Algorithm in Internet-Based Clustering Systems

In-Bok Choi and Jae-Dong Lee

Division of Information and Computer Science, Dankook University,
San #8, Hannam-dong, Yongsan-gu, Seoul, 140-714, Korea
{pluto612, letsdoit}@dku.edu

Abstract. This paper proposes an efficient algorithm for load-sharing and fault-tolerance in Internet-based clustering systems. The algorithm creates a global scheduler based on the Weighted Factoring algorithm. And it applies an adaptive granularity strategy and the refined fixed granularity algorithm for better performance. It may also execute a partial job several times for fault-tolerance. For the simulation, the matrix multiplication using PVM is used in a Internet-based clustering system. Compared to other algorithms such as Send, GSS and Weighted Factoring, the proposed algorithm results in an improvement of performance by 55%, 63% and 20%, respectively. Also, this paper shows that it can process the fault-tolerance.

1 Introduction

Recently, most clustering systems are connected to high-speed network such as Myrinet, SCI, or Gigabit Ethernet for more performance elevation. However, these systems require additional expenses. Also, it is difficult to extend network[1].

With the rapid growth of the Internet, it is easy to build clustering system using computers that are connected to Internet without additional network. However, Internet consists of various networks and heterogeneous nodes. So, there are a lot of possibilities of imbalance and fault by cutting of network and breakdown of nodes. Therefore, in Internet-based clustering systems, a load-sharing algorithm must consider various conditions such as heterogeneity of nodes, characteristics of a network, imbalance of load, and so on.

This paper proposes an efficient algorithm called Efficient-WF algorithm for load-sharing and fault-tolerance in Internet-based clustering systems. The Efficient-WF algorithm creates a global scheduler based on the Weighted Factoring algorithm. Also, it applies an adaptive granularity strategy and the refined fixed granularity algorithm for better performance and executes a partial job several times for fault-tolerance.

The remains of this paper are organized as follows. Section 2 introduces related works about load-sharing algorithms. Section 3 describes the Efficient-WF algorithm.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3036, pp. 18–26, 2004.

Section 4 estimates the performance of the Efficient-WF algorithm. Finally, section 5 states some conclusions and plans for future work.

2 Related Works

Send and GSS algorithm show good performance for load-sharing in NOW(Network of Workstation) environment [3,5]. Weighted Factoring algorithm shows good performance in heterogeneous clustering system [2].

Send algorithm sends the first matrix and t columns of the second matrix together to the first slave, then the first matrix and the next t columns to the second slave, and so on. Each slave multiplies the columns that it receives by the first matrix it already has. When a slave finishes the multiplication, it sends the results back to the master. Master collects the results, writes it into appropriate place of the result matrix and then sends another t columns of the second matrix to the slave. This process continues until all columns of the second matrix have been sent [5].

In GSS(Guided Self-Scheduling) algorithm, a data size scheduled by a function of the remaining data. Given N data and P nodes, the i^{th} data size is determined as follows [3,5]

$$G_i = \left\lceil (1-\frac{1}{P})^i \frac{N}{P} \right\rceil \tag{1}$$

Weighted factoring is identical to factoring except that the size of job in a batch is determined in proportion to the weight(W_j) of nodes. The size of the j^{th} job size in the i^{th} batch is determined as follows [2,3].

$$F_{ij} = \left\lceil (1-\frac{1}{x})^i \frac{N}{x} \frac{W_j}{\sum_{k=1}^{P} W_k} \right\rceil = \left\lceil (\frac{1}{2})^{i+1} N \frac{W_j}{\sum_{k=1}^{P} W_k} \right\rceil \tag{2}$$

3 Design of Efficient-WF Algorithm

This chapter describes the Efficient-WF algorithm for load-sharing and fault-tolerance in Internet-based clustering systems.

3.1 Design of Global Scheduler and Subroutines

A global scheduler is required to apply an adaptive granularity strategy to Weighted Factoring algorithm. Data structure for N data and P slave nodes is as follows.

```
struct  slave_node {
01:   int job[⌊log2N⌋];      // assigned job
02:   int status[⌊log2N⌋];   // 0:to do, 1:doing, 2:done.
03:   float weight;
```

```
04:  int remain;
05:  int doing;
} schedule[P];
```

Send function is usually used when master node assigns job to slave node. At this time, master node achieves following additional works to manage states of slave nodes.

```
Subroutine SEND(schedule[j].job[k], i)
• Input: schedule[j].job[k], received partial result; i, index of a slave node.
01:  send(job of schedule[j].job[k] to i^th node);
02:  schedule[j].status[k] = 1;
03:  schedule[j].remain -= size of schedule[j].job[k];
04:  schedule[j].doing += size of schedule[j].job[k];
End of SEND subroutine
```

Recv function is used when master node gets the partial result from a slave node. Master node also archives following works to manage the states of slave nodes.

```
Subroutine RECEIVE(sched[j].job[k], i)
• Input : schedule[j].job[k], received partial result; i, index of a slave node.
01:  recv(partial result schedule[j].job[k] from i^th node);
02:  schedule[j].status[k]=2;
03:  schedule[j].doing -= size of schedule[j].job[k];
End of RECEIVE subroutine
```

3.2 Load-Sharing Technique by an Adaptive Granularity Strategy

As Weighted Factoring algorithm shares loads by using weight that was evaluated earlier, it is difficult to cope with the change of slave nodes during work. Adaptive load-sharing policy that lowers priority order of slow slave node showed good performance in study of [4].

Therefore, it is desirable to reduce amount of jobs of some slow slave nodes by let some fast slave nodes that finished all their jobs execute the jobs of the slow slave nodes. The adaptive granularity strategy is described as follows.

```
Function AGS_LS(schedule[j].job[k], i)
• Input: schedule[j].job[k], received partial result; i, index of a slave node.
• Output: schedule[m].job[n], next partial job for i^th slave node.
01:  if(schedule[i].remain != 0) {
02:     m = i;
03:     n = k+1;
04:  }
05:  else if(Exist schedule[0…(P-1)].remain != 0) {
```

```
06:     m = index of the slowest slave node that have 'do do' job;
07:     n = the last 'to do' job index of m;
08:   }
09:   return (schedule[m].job[n]);
End of Function AGS_LS(schedule[j].job[k], i)
```

When master node receives a k^{th} partial result of j^{th} slave node from i^{th} slave node, if some jobs remain yet to the i^{th} slave node, master node selects a job according to schedule (line 1-4). If all jobs of the i^{th} slave node are finished (line 5) and 'to do' jobs remain yet in any other slave node (line 6), master node searches the slowest slave node and selects the last job that is not transmitted yet (line 7).

3.3 Load-Sharing Technique by the Refined Fixed Granularity Algorithm

The refined fixed granularity algorithm is to overlap between the time spent by slave nodes on computation and the time spent for network communication [1]. First, master node transmits two jobs in each slave node. Then master node transmits next job to a slave node that transmitted partial result. Therefore, a slave node can achieve next job without waiting for reception of next job from master node. The RFG_LS algorithm that uses the refined fixed granularity algorithm is as follows.

```
Algorithm RFG_LS
• Input: P, number of slave nodes; schedule[P], a scheduled array for P slave
nodes.
• Output: result, merged partial result(e.g. array).
01:   SEND(schedule[0...(P-1)].job[0], 0...(P-1));
02:   SEND(schedule[0...(P-1)].job[1], 0...(P-1));
03:   while(all partial results are not gathered) {
04:     RECEIVE (schedule[j].job[k], i);
05:     schedule[m].job[n] = AGS_LS(schedule[j].job[k], i);
06:     SEND(schedule[m].job[n], i);
07:     MERGE(partial result of schedule[j].job[k]);
08:   }
End of RFG_LS
```

After master node send the first job in each slave node (line 1), send the second job while slave node achieve the first job without receiving partial result (line 2).

3.4 Fault-Tolerance Technique by Executing Jobs Several Times

Clustering systems such as NOW do not guarantee stability of nodes [3]. Algorithms proposed earlier do not consider fault of slave nodes, neither. The reason is that most clustering systems composed into stable regional network environment. However, transmission delay or connection cutting by problems of network can be in Internet.

This paper proposes fault-tolerance technique by executing jobs several times. The technique is that master node assigns ‘doing’ jobs of some slow slave nodes to fast slave nodes that finished all their jobs. This manner is as follows.

Function EJS_FT(schedule[j].job[k], i)
• Input: schedule[j].job[k], received partial result; i, index of a slave node.
• Output: schedule[m].job[n], next partial job for the i^{th} slave node.
01: if(schedule[0…(P-1)].remain==0){
02: m = index of the slowest slave node that have ‘doing’ job;
03: n = the last ‘doing’ job index of m;
04: }
05: return(schedule[m].job[n])□
End of Function EJS_FT(schedule[j].job[k], i)

When master node receives a k^{th} partial result of j^{th} slave node from i^{th} slave node, if all jobs scheduled to i^{th} slave node are finished and any ‘to do’ job is not remain in any other slave node (line 1), master node searches the slowest slave node (line 2) and selects the last ‘doing’ job of the slave node (line 3).

3.5 Efficient-WF Algorithm

The Efficient-WF algorithm extends Weighted Factoring algorithm based on previous paragraph. This Efficient-WF algorithm is as follows.

Algorithm Efficient-WF
• Input: N, size of job; P, number of slave nodes.
• Output: result, merged partial results (e.g. array).
01: CREATE(scheduler by Weighted Factoring algorithm);
02: SEND(schedule[0…(P-1)].job[0], 0…(P-1));
03: SEND(schedule[0…(P-1)].job[1], 0…(P-1));
04: while(all partial results are not gathered){
05: RECEIVE(schedule[j].job[k], i);
06: if(Exist(schedule[0…(P-1)].remain != 0){
07: schedule[m].job[n] = AGS_LS(schedule[j].job[k],i);
08: }
09: else{
10: schedule[m].job[n] = EJS_FT(schedule[j].job[k],i);
11: }
12: SEND(schedule[m].job[n], i);
13: MERGE(partial result of schedule[j].job[k]);
14: }
End of Algorithm Efficient-WF

Master node creates a global scheduler for allocating of jobs (line 1). An adaptive granularity strategy is applied by using AGS_LS function (line 7). And, the Efficient-WF algorithm offers the fault-tolerance by using EJS_FT function (line 10).

3.6 Analysis of Efficient-WF Algorithm

The Efficient-WF algorithm overlaps between the time spent by slave nodes on computation and the time spent for network communication by transmitting previously next job in order to reduce total execution time. In this manner, the time gain is network communication time during slave nodes compute a previous job.

Let (t_s) is the transfer time between master node and slave node. Figure 1. shows an example of the time reduction by overlapping between computation time and network time when the execution times of master node and slave node are equal.

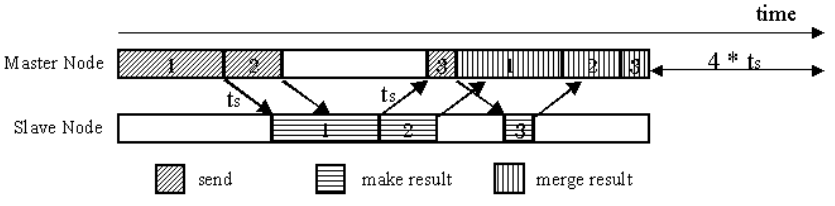

Fig. 1. Time reducing by overlapping between computation time and network time.

Execution time could be reduced by minimum when all network communication time overlap with computing time except the first transfer time from master node to slave node and the last transfer time from slave node to master node. $\lfloor \log_2 N \rfloor$ jobs are assigned to a slave node. Each job is transferred 2 times between master node and slave node. Also, the first transfer time and the last transfer time are excluded. Therefore, for the N data among P slave nodes, the maximum time we can reduce is as follows.

$$t_{profit} = P * ((\lfloor \log_2 N \rfloor * 2t_s) - 2t_s) \quad (3)$$

The Efficient-WF algorithm executes some last jobs several times for fault-tolerance. The slave nodes that have finished all their jobs are considered idle nodes. As these nodes could accomplish faster than the node that has been performing ahead, it could shorten whole execution time as well as process fault-tolerance.

4 Performance Evaluation of Efficient-WF Algorithm

This chapter evaluates the performance of Efficient-WF algorithm that has presented in chapter 3.

4.1 Environment for Performance Estimation

For the performance estimation, total eleven PCs were used. The clustering system composed by one master node and 10 slave nodes. The master node was participated in computing partial results.

Table 1. System configuration (Total 11nodes).

Node Name	CPU	Memory	OS
Master	Pentium3 450	320M	Linux(kernel2.4)
Node1,2	Pentium3 733	128M	Linux(kernel2.4)
Node3,6	Pentium3 450	128M	Linux(kernel2.4)
Node4,5	Pentium3 300	128M	Solaris8.0
Node7	Pentium-pro 133	64M	Linux(kernel2.0)
Node8-9	Pentium-pro 133	32M	Linux(kernel2.0)
Node10	Pentium-pro 133	16M	Linux(kernel2.0)

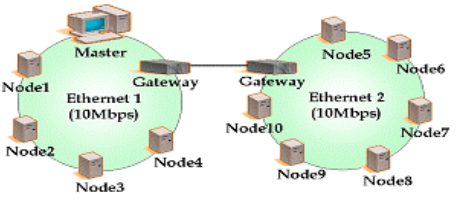

Fig. 2. Network configuration.

4.2 The Result of Performance Estimation

For the simulation, the matrix multiplication using PVM 3.4.4 library is performed on the environment of previous paragraph. All tests are performed 40 times respectively. In this paragraph, WF means the Weighted Factoring algorithm and E-WF means the Efficient-WF algorithm. The Mean value of performance evaluation is as follows.

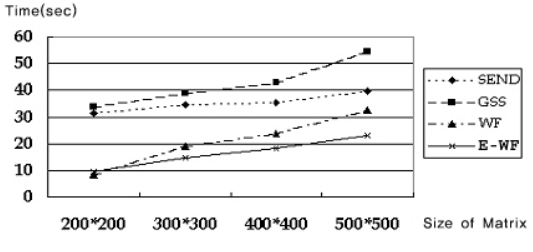

Fig. 3. Performance evaluation by size of matrix

Comparing to other algorithms such as Send, GSS and Weighted Factoring, Effective-WF algorithm results in an improvement of performance by 55%, 63%, 20%, respectively.

To measure fault-tolerance of the Efficient-WF algorithm, sixty-seconds delay is applied in an arbitrary slave node. Sixty seconds is longer than maximum execution time (54.59 seconds) of previous performance evaluation. Applying delay of 60 seconds means that fault happens in the slave node. The result of the test is as follows.

Table 2. Fault-Tolerance. (seconds)

Size of matrix	Fault-Tolerance	Normal state
200*200	9.74	8.75
300*300	15.38	14.49
400*400	18.93	18.38
500*500	23.26	22.81

If the Efficient-WF algorithm cannot offer Fault-Tolerance, the result of the experiment must take more than all 60 seconds. But, we can see that all results are almost equal in Table 4. Although performance of about 5% was fallen than normal state, the result of Table 4 explains that we can get the normal result despite fault happened in slave nodes. Therefore, we can see that the Efficient-WF algorithm can cope with fault of slave nodes.

5 Conclusions and Future Work

This paper has proposed the Efficient-WF algorithm for load-sharing and fault-tolerance in Internet-based clustering systems. The Efficient-WF algorithm uses adaptive granularity strategy and the refined fixed granularity algorithm for load-sharing, and it executes some jobs several times for fault-tolerance.

Comparing to other algorithms such as Send, GSS and Weighted Factoring, the Effective-WF algorithm resulted in an improvement of performance by 55%, 63%, 20%, respectively in experimental clustering system. Also, it offered stable execution time with Fault-Tolerance that could not offer at other algorithms.

The clustering environments in Internet-based clustering systems are more dynamic than current clustering environments. Therefore, adaptive load-sharing techniques in more various environments and compatible fault-tolerance techniques with existing tools will be studied in the future.

Acknowledgements. The present research was conducted by the research fund of Dankook University in 2004. This research was supported by University IT Research Center Project of Korea.

References

1. Bon-Geun Goo, "Refined fixed granularity algorithm on Networks of Workstations", KIPS, Vol.8, No.2, 2001.
2. S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein, "Load-Sharing in Heterogeneous Systems via Weighted Factoring", SPAA, 1997.
3. Yangsuk Kee and Soonhoi Ha, "A Robust Dynamic Load-Balancing Scheme for Data Parallel Application on Message Passing Architecture", PDPTA'98, pp. 974-980, Vol. II, 1998.
4. Jin-Sung Kim and Young-Chul Shim, "Space-Sharing Scheduling Schemes for NOW with Heterogeneous Computing Power", KISS, Vol.27, No.7, 2000.
5. A. Piotrowski and S. Dandamudi, "A Comparative Study of Load Sharing on Networks of Workstations", Proc. Int. Conf. Parallel and Distributed computing system, New Orleans, Oct. 1997.
6. G. Shao, "Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources", Ph.D. thesis, UCSD, June 2001.