Configuration Method of Multiple Clusters for the Computational Grid

Pil-Sup Shin[1], Won-Kee Hong[2], Hiecheol Kim[3], and Shin-Dug Kim[2]

[1] Sungmi Telecom electronics co., LTD., Korea
psshin@sungmi.co.kr
[2] Parallel Processing Lab., Dept. of Computer Science,
Yonsei University, Seoul, Korea
{wkhong, sdkim}@kurene.yonsei.ac.kr
[3] Computer & Communication Eng., Taegu University, Korea
hckim@biho.taegu.ac.kr

Abstract. A Java-Internet cluster platform (JIP) is designed as a computing platform on the computational grid in order to utilize a large collection of computing resources on the Internet. For this goal, a basic cluster module of JIP is defined as a cluster of heterogeneous systems connected to a high-speed network. For a scalable JIP configuration on the computational grid, the basic cluster module can be expanded into a logical set of multiple clusters. JIP is featured with a Java based programming environment, a dynamic resource management scheme, and an efficient parallel task execution mechanism. A multiple cluster configuration is applied to decrease communication time, which is a major bottleneck of performance enhancement. According to the analysis, multiple cluster configuration can enhance the performance of JIP about 2.5 $\sim$ 3 times depending on any application chosen comparing with a single basic cluster configuration.[1]

1 Introduction

The clustering system is considered as a cost-effective system because it is constructed with a lot of commodity systems such as PCs or workstations linked with high speed network [8], [2], [3], [9]. Recently, as the Internet is provided with powerful high-performance machines and the gigabit networks, it can offer powerful infrastructure called computational grid that can solve complex problems, e.g., Legion [5], Globus [4], and Condor [9]. These make it possible to utilize the distributed computing resources on the computational grid for efficient parallel processing.

In this research, a Java-Internet cluster platform (JIP) is designed as a computing platform that makes an efficient link of several computing resources on the computational grid into a single virtual computer to support parallel processing. For this goal, a basic cluster module of JIP is defined as a cluster of

[1] This work was partly supported by 1997 Research Fund from Korea Research Foundation.

various computer systems connected to a high-speed network. For a scalable JIP configuration, the basic cluster module can be expanded into a logical set of multiple clusters. A basic cluster module of JIP is configured as a client, a collection of computing nodes, and a cluster management system (CMS). JIP supports a shared memory system model [1] for communication between any two computing nodes. A configuration method to allow multiple clusters as a single virtual system makes the JIP scalable with ease as the number of participating clusters increases. The JIP is featured with a Java based programming environment, a dynamic resource management scheme, and an efficient parallel task execution mechanism. Based on a prototype implementation of JIP, its configuration method, effectiveness, complexity, and the performance are presented.

The JIP is designed for two primary design issues. The first one is the implementation method of JIP that is designed by Java language [6] and multithreading feature. Java enables heterogeneous clusters as well as computing nodes to be viewed as a single system image and multithreading mechanism allows a simple and effective way to manage computing resources. The second one is the design of a CMS that coordinates the supply and demand for clusters of computing resources. As the CMS plays a role of resource management and resource allocation, JIP can maintain a global view of scalable systems and construct an optimal configuration for high performance.

Simulation is performed to show how the ratio of communication causes any impact on the overall performance of JIP and how the proposed configuration method is effective. It can be shown that a reasonable speedup can be achieved for a single basic cluster module when the number of computing nodes increases especially for data parallel applications. Also the resource management method is evaluated through the numerical analysis of communication time under JIP. Simulation results show that the proposed resource management method can provide the optimal number of clusters for given application characteristics. According to the analysis, the effective number of clusters can be determined depending on the ratio of shared write accesses over the overall shared memory references. Multiple cluster configuration can enhance the performance of JIP about $2.5 \sim 3$ times depending on any application chosen comparing with a single basic cluster configuration.

2 Java-Internet cluster platform

Java-Internet cluster platform (JIP) is a computing environment extending previous distributed parallel computing environment into the computational grid to run parallel applications on multiple clusters at different geographical locations. JIP is designed based on the Java language for platform independence and implemented by using existing mechanisms, namely multithreading and remote method invocation (RMI) mechanisms [6] to create and manage parallelism, which provides a unified access to system resource, called a single system image.

A basic cluster module of the JIP is constructed as a *client*, and a collection of *computing nodes* connected to a high-speed network, and a cluster management

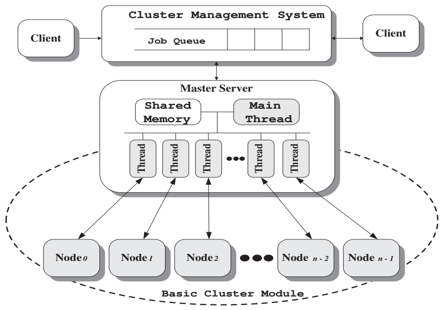


Fig. 1. Basic cluster module of JIP.

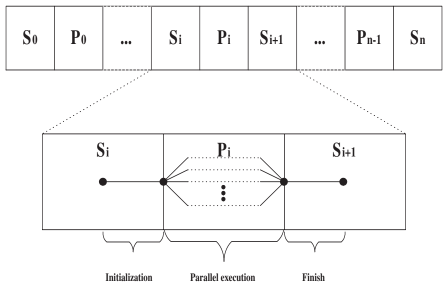


Fig. 2. Programming model for JIP.

system (CMS) as shown in Fig. 1. Client is a system requesting a particular job to run on the JIP and a computing node is a system that provides itself as a computing resource. A cluster is constructed as a master server (host system), and a collection of computing nodes. A master server in the cluster plays a role as a master control system, composed of a portion of shared memory for communication among computing nodes. CMS coordinates the supply and demand for computing resources and is in charge of determining a specific configuration of clusters. Because CMS also includes a job queue, multiple job requests can be processed sequentially. For a given Java application, a basic cluster module can be selected by explicit programming under configuration guideline given by the JIP. Then the CMS selects the master server as a control system to coordinate computing nodes of the selected cluster module and sends a Java parallel program. Then, the master server starts to execute a given Java application over this single virtual machine. A requested Java application is run on the top of master server and a number of threads using Java multithreading are created, where each thread is associated with one computing node. Specifically each thread is in charge of executing a block of Java code and data, called as a Java code block, on a computing node.

In the JIP, a collection of computing nodes is determined according to the proposed configuration method and its corresponding number of clusters to attain a reasonable performance gain considering communication overhead is determined. This configuration method can be expanded as a hierarchical configuration of multiple clusters to support scalability.

2.1 Effective parallelism on JIP

As shown in Fig. 2, an application program can be represented as a set of Java code blocks under the JIP programming model, where code blocks can be divided into two types, serial code blocks (S_i) and parallel code blocks (P_i). For any chosen basic cluster module the corresponding master server will execute the serial code blocks, where parallel code blocks are executed by computing nodes within that cluster. Also the master server is in charge of controlling the overall program execution. The programmer may specify the boundary of S_i and P_i explicitly at programming time. Two code blocks, S_i and P_i, are interleaved for all i. For a given application, its corresponding system configuration is determined based on some algorithmic parameters.

In general, parallelism can be classified into function parallelism and data parallelism. A set of three code blocks, i.e., S_i, P_i, and S_{i+1}, is called as a *basic parallel module*. Specifically, applications based on function parallelism can be formed as one basic parallel module or a small number of basic parallel modules. For data parallelism, a collection of basic parallel modules is usually executed on a fixed set of computing nodes. The JIP can execute efficiently function parallelism as well as data parallelism. For those applications based on the function parallelism, the number of computing nodes can be determined by the number of independent or cooperating sub-tasks explicitly at programming time. In the case of data parallelism, programmers can specify parallelism explicitly based on a configuration method that can guide to select some system configuration parameters, i.e., the optimal number of clusters and computing nodes allocated per cluster. If the performance gain by additional parallelism is smaller than the overhead resulted from the network traffic incurred for parallel execution, the overall performance gain cannot be attained. Thus system configuration parameters should be chosen based on the trade-off between additional parallelism and the overhead incurred.

2.2 Configuration of multiple clusters

According to the degree of parallelism for any application, tens or hundreds of computing nodes may be used. If all computing nodes should be connected to a single master server to communicate with other computing nodes, a moderate amount of delay time will be occurred per communication and significant bottleneck problem will be caused at this centralized master server. To overcome this problem, JIP is designed for supporting multiple cluster configuration. According to the characteristics of algorithms and the patterns of shared memory references in parallel applications [7], the number of clusters is determined. Each cluster consists of one master server and multiple computing nodes.

If all the computing nodes joined for a particular job belong to several clusters, data can be classified into three types, i.e., private, locally shared, and globally shared. The private data can be accessed only by those dedicated computing nodes within a cluster. Locally shared data are shared only by a set of computing nodes belonging to a specific cluster. Globally shared data are shared among a set of chosen clusters. Thus, JIP supports a *local shared memory (LSM)* for efficient communication within a cluster. LSM is a block of shared memory allocated for each cluster and accessed by only computing nodes in the given cluster. A conventional shared memory space that can be accessed by all joined computing nodes is named as a *global shared memory (GSM)* to be distinguished from the LSM.

While one computing node is accessing its LSM to update data, other computing nodes in the same cluster should wait to access the LSM. However, the

Table 1. Variables used in numerical analysis.

Variable	Meaning	Variable	Meaning
S_{PD}	Size of Private Data	S_P	Size of a packet in unit words
S_{SD}	Size of Shared Data	N	Number of computing nodes
R_M	Ratio of SM reference instructions	G	Number of clusters
R_{MW}	Ratio of write-updates to SM references	T_{MR}	Ratio of read to SM
R_{MWL}	Ratio of LSM writes over all SM writes	R_{MWG}	Ratio of GSM writes over all SM writes
T_{SP_i}	Sequential execution time for i-th code	A	Multicasting Coefficient in master server
T_{UL}	Unit word reference time at master server	T_{UG}	Unit word reference time at CMS

other computing nodes contained in the other clusters can access their own LSM or the GSM in parallel without any delay. While a datum of GSM is updated, no computing node joined in processing can access the GSM until the current update is completed. For read-only references, LSM and GSM are accessed in the same way and all computing nodes can read the shared data independently at the same time. For write references, its procedure becomes different depending on the target, i.e., LSM or GSM. Because computing nodes in the same cluster share the LSM, only the computing nodes in the same cluster should wait until any update is completed. For the case of accessing GSM, all computing nodes should wait until the GSM is unlocked and the communication is finished to keep data coherency among GSMs in all master servers.

2.3 Global view of JIP

To accommodate a lot of users in a wide area cluster environment, CMS is very important. The CMS manages multiple clusters as one's own global virtual machine. In JIP, a client can construct a hierarchical and scalable system configuration via CMS. When a client requests a job to the CMS, the client transfers the Java parallel program with a header including the information of algorithmic characteristics, such as the degree and type of parallelism and the ratio of computation to communication. The CMS uses this information and decides the optimal number of computing nodes for achieving the optimal overall performance and eventually determines the number of clusters. For this, we must consider a programming model of JIP addressed in prior section.

As shown in Fig. 2, when T_{S_i} and T_{P_i} denote the times taken to execute the i-th sequential code block (S_i) and the i-th parallel code block (P_i) respectively in the program model, the total execution time (T) for a parallel program can be defined as the following equation:

$$T = \sum_{i=0}^{n} T_{S_i} + \sum_{i=0}^{n-1} T_{P_i}. \tag{1}$$

Sequential code blocks (S_i) are executed by the master server and parallel code blocks (P_i) are executed by computing nodes. Except both the first and last sequential code blocks, T_{S_i} is divided into the three parts, $T_{S_i}^G$, $T_{S_i}^D$, and

$T^A_{S_i}$, which are the time to gather the execution results of the prior parallel code block (P_{i-1}) from the computing nodes, the time to dispatch code and data for parallel execution of the posterior parallel code block (P_i), and the time to execute remaining pure sequential code respectively. The first sequential code block (S_0) does not include the time to collect the results produced by the prior parallel code block and the last sequential code block (S_n) does not require any time to prepare for the posterior parallel code block, either. T_{P_i} is defined as the sum of $T^M_{P_i}$ and $T^E_{P_i}$, which are the times taken to execute all the shared memory reference code and the other parallel code respectively in the i-th parallel code block.

$$T_{S_i} = \begin{cases} T^A_{S_0} + T^D_{S_0}, & \text{if } i = 0 \\ T^G_{S_i} + T^A_{S_i} + T^D_{S_i}, & \text{if } 0 < i < n \\ T^G_{S_n} + T^A_{S_n}, & \text{if } i = n \end{cases}$$
$$T_{P_i} = T^M_{P_i} + T^E_{P_i}. \tag{2}$$

Thus, Eq.(1) can be changed as follows:

$$T = \sum_{i=0}^{n} T^A_{S_i} + \sum_{i=0}^{n-1} T^E_{P_i} + \sum_{i=0}^{n-1} T^C_i, \quad T^C_i = (T^D_{S_i} + T^M_{P_i} + T^G_{S_{i+1}}). \tag{3}$$

According to Eq.(3), T can be classified into the execution times of a master server and computing nodes, and the three types of communication times (T^C_i) such as dispatching time, shared memory accessing time, and gathering time. To clarify those communication times in terms of low-level communication parameters, several terms are defined as in Table 1. If data are transferred by the packet basis in the high-speed network, a burst transfer of data is preferred. $T^D_{S_i}$ and $T^G_{S_i}$ can be obtained as follows:

$$T^D_{S_i} = \left\lceil \frac{S_{PD} + S_{SD}}{S_P \cdot N} \right\rceil \cdot T_U, \quad T^G_{S_i} = \left\lceil \frac{S_{SD}}{S_P \cdot N} \right\rceil \cdot T_U. \tag{4}$$

For any P_i, the parallel code as well as the shared memory accessing code are assumed to be distributed equally among all the computing nodes for simplicity. For a given P_i, all the computing nodes can perform parallel code allocated concurrently, except for the shared memory accesses that should be processed sequentially at the master server. Thus $T^E_{P_i}$ and $T^M_{P_i}$ can be obtained as follows:

$$T^E_{P_i} = (1 - R_M) \cdot \frac{T_{SP_i}}{N},$$
$$T^M_{P_i} = R_M \cdot T_{SP_i} \cdot \left(\frac{1 - R_{MW}}{N} + R_{MW} \right) \cdot A_N. \tag{5}$$

The term of A_N in Eq.(5) is multicasting coefficient increasing according to the number of computing nodes connected to one master server, where A_N can represent $k \cdot N$, the ratio of the time to send the same Java code block to multiple computing nodes over the time to send to a single computing node. This equation is obtained from the simulation using *smpl* and *ether* model.

For obtaining the optimal number of computing nodes (N), the execution time of parallel code block, Eq. (5), must be minimized. This equation is as follows.

$$MIN\,(T_{P_i}(N)) = MIN\,\left(T^{E}_{P_i} + T^{M}_{P_i}\right) = MIN\{(1 - R_M)\cdot \frac{T_{SP_i}}{N} + R_M \cdot T_{SP_i} \cdot \left(\frac{1-R_{MW}}{N} + R_{MW}\right)\cdot A_N\}. \quad (6)$$

The execution time of parallel application according to the number of participating computing nodes (N) is varied depending on the ratio of total shared memory references and the ratio of write references among shared memory references as Eq.(6). To obtain the optimal number of computing nodes as the execution time of a parallel application, Eq.(6) is changed into equation with respect to N. The equation is differentiated by N for getting an optimal number of computing nodes. Therefore, decision function is derived as Eq.(7).

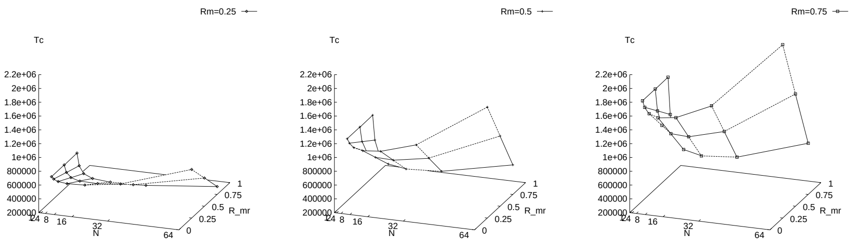
$$T'_{P_i}(N) = 0, \qquad N = \sqrt{\frac{(1-R_M)\cdot T_{SP_i}}{R_M \cdot T_{SP_i} \cdot R_{MW} \cdot k}},\ \mathrm{N} > 0\ . \quad (7)$$

After the CMS determines the number of optimal computing nodes according to the information of algorithmic characteristics of application, it checks if the number of available computing nodes in multiple clusters is enough. If the number of available computing nodes in a cluster is smaller than the optimal number, the CMS constructs a multiple cluster configuration of JIP. For this, the CMS also needs to have a decision function which can decide the number of clusters for optimal performance in the multiple cluster configuration of JIP. If it is assumed that the amount of references to the LSM as well as the GSM is distributed equally among all the computing nodes, $T^{M}_{P_i}$ is changed as follows.

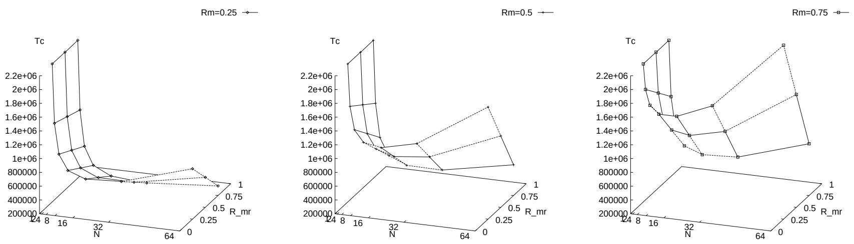
$$T^{M}_{P_I} = R_M \cdot T_{SP_i} \cdot \{\left(\frac{1-R_{MW}}{N} + \frac{R_{MW}\cdot R_{MWL}}{G}\right)\cdot \frac{A_N}{G} + R_{MW}\cdot R_{MWG}\cdot T_{UG}\cdot G\}. \quad (8)$$

For obtaining the optimal number of clusters (G), the execution time of parallel code block, Eq.(5), must be minimized. Thus, this equation is changed as follows.

$$\begin{aligned} MIN\,(T_{P_i}(G)) &= MIN\,\left(T^{E}_{P_i} + T^{M}_{P_i}\right) \\ &= MIN\Big(R_M \cdot T_{SP_i} \cdot \{\left(\frac{1-R_{MW}}{N} + \frac{R_{MW}\cdot R_{MWL}}{G}\right)\cdot \frac{A_N}{G} \\ &\quad + (1-R_M)\cdot \frac{T_{SP_i}}{N} + R_{MW}\cdot R_{MWG}\cdot T_{UG}\cdot G\}\Big). \end{aligned} \quad (9)$$



(a) Communication time (T_i^C) according to R_M.



(b) Overall execution time (T_{P_i}) according to R_M.

Fig. 3. Numerical analysis of a basic cluster module of JIP.

If the Eq.(9) is arranged with respect to G and differentiated by G, we can get the optimal number of clusters. Therefore, when $T'_{P_i}(G) = 0$, decision function for the optimal number of clusters is shown in the Eq.(10).

$$\begin{aligned}
A &= \frac{1}{2} \cdot \left\{ \frac{2 \cdot R_{MWL} \cdot A_N}{R_{MWG} \cdot T_{UG}} \right. \\
&\quad \left. + \sqrt{\left(\frac{-2 \cdot R_{MWL} \cdot A_N}{R_{MWG} \cdot T_{UG}}\right)^2 + 4 \cdot \left(\frac{-(1 - R_{MW}) \cdot A_N}{3 \cdot R_{MW} \cdot R_{MWG} \cdot T_{UG} \cdot N}\right)^3} \right\}, \\
B &= \frac{1}{2} \cdot \left\{ \frac{2 \cdot R_{MWL} \cdot A_N}{R_{MWG} \cdot T_{UG}} \right. \\
&\quad \left. + \sqrt{\left(\frac{-2 \cdot R_{MWL} \cdot A_N}{R_{MWG} \cdot T_{UG}}\right)^2 - 4 \cdot \left(\frac{-(1 - R_{MW}) \cdot A_N}{3 \cdot R_{MW} \cdot R_{MWG} \cdot T_{UG} \cdot N}\right)^3} \right\}, \\
G &= \sqrt[3]{A} + \sqrt[3]{B}, \ \mathrm{G} > 0 \ . \qquad (10)
\end{aligned}$$

After determining the optimal number of clusters, the CMS constructs a multiple cluster configuration as many as dividing the optimal number of computing nodes into the number of clusters.

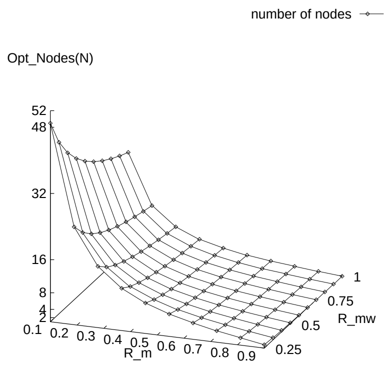


Fig. 4. Optimal number of computing nodes.

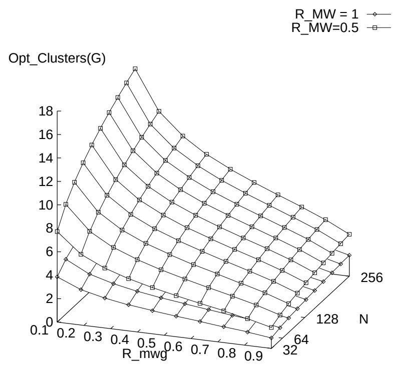


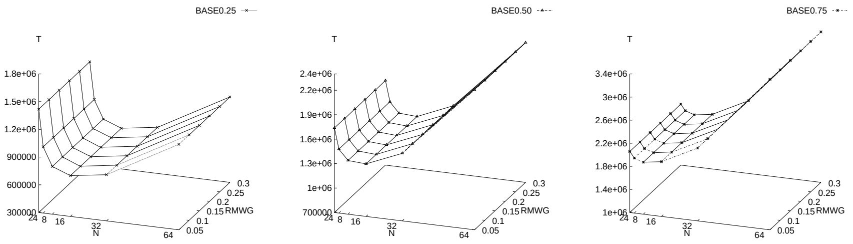
Fig. 5. Optimal number of clusters.

3 Performance evaluation

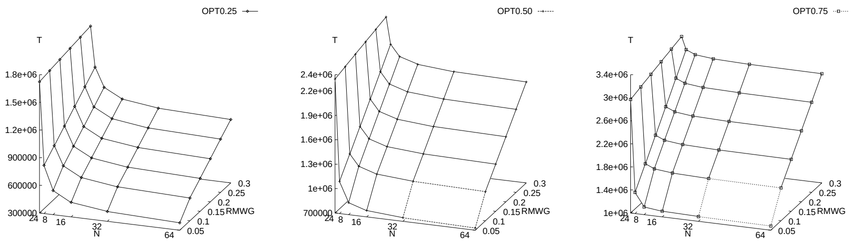
In this section, we simulate the performance of JIP constructed with a basic cluster module or multiple clusters by using the previous equations. Fig. 3 depicts the performance of a basic cluster module of JIP using Eq.(4) and (5), when $N = 1 \sim 64$. Fig. 3 (a) shows the variation of communication time, T_i^C, according to the ratio of shared memory read references and the y-axis represents the time in milli-seconds(ms). It says that if $N \geq Opt\_N$, T_i^C does not decrease any more, but increases as the number of computing nodes increases. It is derived from the increased network contention at a master server. Fig. 3 (b) shows the variation of overall execution time, T. The ratio of performance speedup is decreased more and more as a program includes higher R_M until N becomes Opt_N, because of communication overhead that is increased suddenly, when $N \geq Opt\_N$. It means that effective parallelism and the scalability of given program vary according to R_M.

Fig. 4 depicts the optimal number of computing nodes according to characteristics of application, such as the ratio of shared memory references (R_M) and the ratio of shared memory write references (R_{MW}), obtained by using Eq.(7). According to Fig. 4, the optimal number of computing nodes, N, is decreased as R_M and R_{MW} are increased, because communication overhead occurred at a master server is increased as the R_M and R_{MW} are increased. Therefore, if the CMS constructs basic cluster module with optimal number of computing nodes by using Eq.(7), each client participating in JIP can be guaranteed by the optimal performance.

Fig. 5 depicts the optimal number of clusters according to the number of participating computing nodes (N) and the ratio of global shared memory write references (R_{MWG}), when $R_{MW} = 1$ and $R_{MW} = 0.5$, by using Eq.(10). According to Fig. 5, the optimal number of clusters, G_{OPT}, is increased as R_{MW} and



(a) Overall execution time in a basic cluster configuration.



(b) Performance by multiple clusters.

Fig. 6. Performance comparison of a basic model with scalable model.

R_{MWG} are decreased and N is increased. As R_{MWG} is increased, the overhead derived from data coherency procedure exceeds the gain by parallel memory writes and G_{OPT} becomes small. When N is increased, contention at the master server is also increased and memory latency becomes longer. It means that G_{OPT} is increased as N increases.

Fig. 6 shows the performance comparison for a basic cluster module with a multiple cluster configuration when the values of R_M are 0.25, 0.5, and 0.75. Fig. 6 shows that the performances of basic cluster module and multiple cluster configuration are depicted by comparing the OPT as a multiple cluster configuration with the BASE as the basic cluster module. The number in model, e.g., OPT0.25 indicate the value of R_M. As shown in this figure, in the case of $N = 2$, the performance of multiple cluster configuration gets worse because the preparation time and coherent time is increased. In all cases of R_M, if $N \geq 4$ and $R_{MWG} \leq 0.15$, multiple cluster configuration shows the better performance than the base cluster module and if $N \geq 64$ and $R_{MWG} \leq 0.3$, multiple cluster configuration shows the better performance than the base cluster module. In the case of $R_{MWG} = 0.05$, multiple cluster configuration can enhance the performance by $2.5 \sim 3$ times. The performance approaches near to the point of the basic cluster module as R_{MWG} is increased. If $R_{MWG} > 0.3$, multiple cluster configuration causes significant performance degradation. It is due to the increased overhead from the process of data coherency.

4 Conclusion

For building an efficient cluster computing platform using geographically distributed computing resources on the grid, a Java-Internet cluster platform is designed in the aspects of high programmability, efficient management for a cluster of computing nodes, system scalability, and code compatibility. The JIP is implemented with Java language and multithreading mechanisms, which supports a shared memory system with sequential consistency model. Performance of multiple clusters is evaluated through the analysis of communication time under JIP, and verified through the simulation. According to the analysis, the effective number of clusters can be determined depending on the ratio of shared memory write references. Specifically, multiple cluster configuration can enhance the performance of JIP about $2.5 \sim 3$ times in those applications with a reasonably high ratio of shared memory references.

References

1. Adve, S.V., Gharachoroloo, K.: Shared Memory Consistency Models : A Tutorial. IEEE Computer (1996) 66-76.
2. Bal, H.E., Plaat, A., Kielmann, T., et. al.: Parallel Computing on Wide-Area Clusters: the Albatross Project. Proc. of Extreme Linux Workshop (1999) 20-24.
3. Culler, D.E., Arpaci-Dusseau, A., Arpaci-Dusseau, R., Chun, B., Lumetta, S., Mainwaring, A., Martin, R., Yoshikawa, C., Wong, F.: Parallel Computing on the Berkeley NOW. Joint Symposium on Parallel Processing (1997) 30-40.
4. Foster, I., Kesselman, C.: The Globus Project: A Status Report. IPPS/SPDP'98 Heterogeneous Computing Workshop (1998) pp.4-18.
5. Grimshaw, A.S., Wulf, W.A., the Legion team: The Legion Vision of a Worldwide Virtual Computer. Communications of the ACM, Vol. 40(1) (1997).
6. Java Remote Method Invocation Specification. Sun Microsystems, Mountain View, California (1997).
7. Pinkston, T.M., Baylor, S.J.: Parallel Processor Memory Reference Analysis: Examining Locality and Clustering Potential. 5th SIAM Conference on Parallel Processing for Scientific Computing (1992) 513-518.
8. Raman, A., Rajkumar, et. al.: PARDISC: A Cost Effective Model for Parallel and Distributed Computing. Proc. of Int. Conf. on High Performance Computing, IEEE Press (1996).
9. Tannenbaum, T., Litzkow, M.: The Condor Distributed Processing System. Dr.Dobbs' Journal, Vol. 20 (1995) 42-44.